Web Service Discovery Based on Semantic Information

Query Formulation and Adaptive Matchmaking

Vom Fachbereich Elektrotechnik und Informationstechnik
der Technischen Universität Darmstadt
zur Erlangung des akademischen Grades eines Doktor-Ingenieurs (Dr.-Ing.)
genehmigte Dissertationsschrift von
Dipl.-Oec. Stefan Schulte, B.Sc., MIT (University of Newcastle) aus Meppen
2010 – Darmstadt – D17

TECHNISCHE
UNIVERSITÄT
DARMSTADT

Fachbereich Elektrotechnik
und Informationstechnik

Fachgebiet Multimedia Kommunikation
Prof. Dr.-Ing. Ralf Steinmetz

Web Service Discovery Based on Semantic Information
Query Formulation and Adaptive Matchmaking

Vom Fachbereich Elektrotechnik und Informationstechnik der Technischen Universität Darmstadt
zur Erlangung des akademischen Grades eines Doktor-Ingenieurs (Dr.-Ing.)
genehmigte Dissertationsschrift von
Dipl.-Oec. Stefan Schulte, B.Sc., MIT (University of Newcastle) aus Meppen.

Tag der Einreichung: 20. April 2010
Tag der Disputation: 11. Juni 2010

Vorsitz: Prof. Dr.-Ing. Helmut F. Schlaak
Erstreferent: Prof. Dr.-Ing. Ralf Steinmetz
Korreferent: Prof. Dr. York Sure

Technische Universität Darmstadt
Fachbereich Elektrotechnik und Informationstechnik

Fachgebiet Multimedia Kommunikation (KOM)
Prof. Dr.-Ing. Ralf Steinmetz

Abstract

Service-oriented Computing introduces a range of possible applications spanning from the combination of Web services in software mashups to the design and implementation of entire IT system landscapes following the paradigm of Service-oriented Architectures. The discovery of services which provide a desired capability is one of the basic operations in Service-oriented Computing and is deemed to be one of the grand challenges in Web service research. This applies in particular to scenarios with a large number of service offers, where it is desirable to automate the discovery process to some degree.

Service discovery is based on the description of service components, e.g., interfaces or operations. As the syntactic description of a Web service is often imprecise, semantic Web services are considered to play a decisive role in the facilitation of service discovery. In this context, the application and utilization of semantic information in service discovery concerns the ability of service providers to describe services, the ability of requesters to specify requirements, and the effectiveness of the service matchmaker, i.e., an algorithm that takes into account a request and finds the best fitting services from a set of service offers. Matchmaking research focuses on the selection of the necessary elements from a service description, similarity metrics, and the combination of the resulting similarity values. This thesis provides several contributions to the improvement and ease of service discovery based on semantic information. The main contributions are made in the fields of service matchmaking and query formulation.

Regarding the first-mentioned contribution, two approaches to matchmaking for semantic Web services are presented. The first of which, LOG4SWS.KOM, is based on "classic" subsumption matching and introduces an innovative way to weight and combine different matching degrees. LOG4SWS.KOM is self-adaptive to different basic assumptions regarding the semantic concepts applied in a service description. This includes different presumptions regarding what a semantic annotation on a distinct service abstraction level actually denotes as well as the meaning of different subsumption relationships between semantic concepts. LOG4SWS.KOM is applied to different abstraction levels of a service description, which may not necessarily be completely described using semantic information. Hence, the matchmaker includes a linguistic-based fallback strategy, triggering the need to incorporate syntactic information. The second matchmaker, COV4SWS.KOM, deviates from logic-based similarity measurement and applies methods from the field of relatedness measurement of semantic concepts in ontologies. This way, COV4SWS.KOM allows more fine-grained relationships than conventional subsumption matching-based approaches. Additionally, COV4SWS.KOM introduces the adaptation to varying quality and usefulness of syntactic descriptions and semantic annotations at different abstraction levels of a service description. Both matchmakers are implemented for SAWSDL and provide, to the best of our knowledge, the best matchmaking results for this Web service standard regarding Information Retrieval metrics, so far.

Regarding the second focus of this thesis – query formulation for semantic Web service discovery – an extensive analysis of requirements towards a unified service query formalism has been conducted. Based on this analysis, two different approaches to query formulation for semantic Web services have been designed, developed, and implemented. The first is a lightweight approach making use of already existing standards and technologies: Here, a slightly extended SPARQL syntax for SAWSDL-based service descriptions is integrated into UDDI. However, the usage of existing standards imposes some constraints, as especially SPARQL has not been explicitly designed for query formulation for semantic Web services. Hence, a second, more advanced approach, has been implemented, where a distinct, SPARQL-based query language is conceptualized and integrated in a service registry. This language – SWS2QL – allows a service requester to address different service abstraction levels, incorporate and parameterize matchmakers, define thresholds, etc., leading to a sophisticated, fine-grained definition of service requests. Even though the corresponding proof of concept implementation makes use of ebXML as service registry standard and SAWSDL as service formalism, results can be easily transferred to other registry and service technologies, as the approach is based on abstract service data and query models. This way, a unified service query formalism is provided.

Apart from the main contributions, this thesis also provides a general framework based on ebXML, which features the integration of semantic Web service descriptions and different service matchmakers into this registry standard.

Kurzfassung

Mögliche Anwendungen Serviceorientierter Konzepte reichen von der Verknüpfung von Webapplikationen in Mashups bis hin zur Gestaltung komplexer IT-Anwendungslandschaften in Form von Serviceorientierten Architekturen. Dabei kommt dem Auffinden von Diensten eine hohe Bedeutung zu. Insbesondere in Szenarien, in denen eine Vielzahl von Diensten zur Verfügung steht, ist zumindest ein gewisser Grad an Automatisierung des Suchprozesses wünschenswert und letztendlich notwendig, um darauf aufbauende Funktionalitäten wie beispielsweise die Ad-hoc-Einbindung von Diensten zu ermöglichen.

Das Auffinden von Diensten basiert auf der Beschreibung einzelner Dienstelemente wie Schnittstellen oder Operationen. Gerade bei der automatischen Einbindung von Diensten werden genaue und maschinenverarbeitbare Angaben benötigt, welche über eine syntaktische Darstellung hinausgehen. Aus diesem Grund werden semantische Informationen genutzt, um Dienste präzise zu spezifizieren. Diese Informationen können dann in der Dienstsuche verwendet werden, um Anforderungen an einen Dienst möglichst genau zu beschreiben. Ein Matchmaker, d. h. ein Algorithmus, der in der Lage ist, basierend auf einer Anfrage die am besten passenden Dienste aus einer Angebotsmenge zu bestimmen, muss dementsprechend in der Lage sein, semantische Informationen verarbeiten und nutzen zu können. Die vorliegende Arbeit liefert verschiedene Beiträge zur Vereinfachung und Verbesserung der Suche nach funktional passenden Diensten durch die Entwicklung und Umsetzung innovativer Matchmaker und Anfragesprachenkonzepte.

Der erste in dieser Arbeit vorgestellte Matchmaker – LOG4SWS.KOM – basiert auf der Deduktion von Inferenzrelationen zwischen semantischen Konzepten. Dieser Ansatz wird um ein innovatives Verfahren zur Gewichtung und Kombination einzelner Ähnlichkeitswerte erweitert. LOG4SWS.KOM ermöglicht die Adaption an verschiedene Grundannahmen bezüglich der verwendeten semantischen Konzepte und passt sich entsprechend an verschiedene Dienstdomänen an. Weiterhin werden in LOG4SWS.KOM unterschiedliche Abstraktionsebenen von Diensten berücksichtigt, welche nicht zwangsläufig vollständig semantisch beschrieben sind. Aus diesem Grund verfügt der Matchmaker über eine alternative Methodik zur Bestimmung von Ähnlichkeiten zwischen einzelnen Dienstelementen, welche syntaxbasiert ist. Der zweite Matchmaker – COV4SWS.KOM – weicht von der üblicherweise verwendeten Logik-basierten Bestimmung von Ähnlichkeitsmaßen ab. Stattdessen werden Vergleichsmaße für Ontologien verwendet, welche die Ähnlichkeit zwischen semantischen Konzepten in einer feineren Granularität als logische Verfahren bestimmen. Zusätzlich ermöglicht COV4SWS.KOM die Adaption an die unterschiedliche Güte und Nutzbarkeit von Beschreibungen auf verschiedenen Dienstebenen. Beide Matchmaker wurden für SAWSDL umgesetzt und liefern die bisher besten Suchergebnisse für diesen Beschreibungsstandard bezüglich bekannter Gütemaße aus dem Bereich des Information Retrieval.

Hinsichtlich des zweiten inhaltlichen Schwerpunkts dieser Arbeit – der Formulierung von Anfragen für semantisch beschriebene Dienste – wurden zunächst auf konzeptioneller Ebene Anforderungen an einen Suchformalismus für Dienste erarbeitet. Diese umfassen die Kombination von syntaktischen und semantischen Informationen, die Definition von Wertebereichen (Zielmengen) sowie die Anpassung des Matchmakings durch den Nutzer. Als Machbarkeitsnachweis wird das Anfragekonzept in einer leichtgewichtigen Lösung für UDDI und in einer umfangreicheren Lösung für ebXML umgesetzt. Im ersten Fall kommen bereits existierende Standards wie SAWSDL, ein offizielles Mapping von SAWSDL nach RDF und eine leicht erweiterte Version von SPARQL als Anfragesprache zum Einsatz. Es zeigt sich, dass die Wiederverwendung von Standards einige Einschränkungen mit sich bringt, da insbesondere SPARQL nicht als Anfragesprache für semantisch beschriebene Dienste konzipiert wurde. Daher wird in der zweiten Lösung zunächst von existierenden Technologien abstrahiert und SPARQL wird als Grundlage für eine neue, auf Dienste abzielende Anfragesprache – SWS2QL – verwendet. SWS2QL ermöglicht dem Anwender die explizite Adressierung unterschiedlicher Serviceabstraktionsebenen, die Definition von Wertebereichen bzw. Mindestähnlichkeiten und die Parametrisierung von Matchmakern innerhalb einer Anfrage. Auf diese Weise kann ein Nutzer seine Anforderungen an einen Dienst sehr genau und feingranular definieren. Obwohl sich die entsprechende Beispielimplementierung auf SAWSDL und ebXML bezieht, kann der Ansatz sowie die Anfragesprache aufgrund der verwendeten abstrakten Service- und Anfragemodelle auf andere Standards übertragen werden.

Als zusätzlicher Beitrag wird in dieser Arbeit ein Framework zur Integration von semantisch beschriebenen Diensten sowie unterschiedlichen Matchmakern in ebXML präsentiert.

Acknowledgments

The research presented in this thesis has been conducted during my time as a research assistant at the Multimedia Communications Lab (KOM) at Technische Universität Darmstadt, Germany, under the supervision of Prof. Dr.-Ing. Ralf Steinmetz. During the last four years, a number of different people have contributed to the success of my doctoral studies:

Special thanks go to my colleagues from the "Service-oriented Computing" research group – my current colleagues Dr.-Ing. Julian Eckert, Ulrich Lampe, André Miede, Michael Niemann, Apostolos Papageorgiou, Dieter Schuller, and Melanie Siebenhaar, as well as my former colleagues Dr.-Ing. Rainer Berbner and Dr.-Ing. Nicolas Repp. Without your support and the excellent working atmosphere, it would not have been possible for me to finish this thesis.

Furthermore, I'd like to thank all former and current colleagues at KOM, especially Prof. Dr.-Ing. Markus Fidler, Dr.-Ing. Kalman Graffi, Prof. Dr.-Ing. Matthias Hollick, Sebastian Kaune, Dr.-Ing. Aleksandra Kovačević, and Andreas Reinhardt, for their very useful advice and the constructive discussions. In addition, I would like to thank everybody who proof-read my dissertation.

Prof. Dr.-Ing. Ralf Steinmetz gave me the opportunity to work in his lab and to participate in a number of very interesting research projects. Without his constant support and advice, I would be still very far away from completing this thesis. Also, I would like to thank my co-supervisor Prof. Dr. York Sure from the University of Koblenz-Landau for his help, especially while finishing my work.

A special thanks goes to my parents and my brother Markus, who supported me so much in my education and my life in general. Thank you for giving me the chance to see the world and go my own way. Furthermore, I want to thank my friends Korbinian von Blanckenburg, Mira Diekmann, Stefan Focks, and Achim Gebhardt for bringing me back down to earth whenever necessary.

Finally, I would like to thank you, Nora, for being part of my life and all your support.

Darmstadt, 2010

Contents

Part I
Introduction and Background

1 Introduction

The need for information technologies which are able to support agile organizations and fast-changing business processes, has led to the wide propagation of Service-oriented Computing (SOC). Today, SOC is a multi-level approach, ranging from the engineering and operation of IT infrastructures to the usage in small Web-based applications called software mashups. One particular application area, which has heavily influenced the computer science research community, as well as the software industry, in recent years is the Service-oriented Architecture (SOA) paradigm, where services are deployed in order to organize and implement IT architectures and, eventually, realize Business/IT alignment.

Independent of the actual application area, SOC is based on services. Services are self-describing encapsulations of functionalities offered by software components. As services are loosely coupled and self-contained, it is possible to dynamically invoke and substitute services, e.g., in a business process, even across the borders of a single company or organization [148]. Hence, one particular application area is the usage of services in workflows, i.e., IT-supported business processes.

Many use cases for SOC are based on the invocation of services through the Internet using Web service standards like the Web Service Description Language (WSDL) and the Simple Object Access Protocol (SOAP). In fact, Web service technologies are currently the most common way to implement service-oriented concepts and have led to the vision of an "Internet of Services", where services for all areas of life and business will be offered on the Web [197, 258]. Regardless of whether using services on a large or small scale, the discovery of services, which offer a distinct functionality under certain non-functional requirements is one of the vital steps in service invocation and is hence deemed one of the grand challenges in Web service and SOA research [208, 259]. For the discovery of Web services, it is necessary to describe the offered and requested services' elements and capabilities as precisely as possible. Ideally, a service consumer would find a range of ready-to-use services, which are offered by different service providers.

Today's Web service standards are primarily motivated by the interoperability of software components over the Web and rely on the Extensible Markup Language (XML) [258]. As there is no further framework for the accurate description of service functionalities and properties, services are usually described using natural language which is often too imprecise. As a result, WSDL-based service descriptions cannot be interpreted without human intervention. Accordingly, further functionalities such as automated Web service discovery, execution, or composition are very difficult to achieve [243]. However, a certain degree of automation is necessary in order to achieve the application of Web service technologies on a larger scale [144].

In order to overcome the shortcomings of syntax-based service descriptions, several researchers have proposed the usage of semantic information in Web services, resulting in the concept of semantic Web services (SWS). Here, it seems reasonable to apply methods, tools, and technologies from the *Semantic Web*, which is an extension of the current Web as envisioned by Berners-Lee [27]. The comprehensible Semantic Web activities of the World Wide Web Consortium (W3C) aim to augment information on the Web with a well-defined meaning and provide a layer of machine-interpretable data. The ultimate goal is to accomplish tasks automatically by supplying machines with adequate information.

Today, SWS are a prominent field of research and have resulted in a number of different approaches and standards such as the Web Ontology Language for Web Services (OWL-S), the Web Service Modeling Language (WSML), or Semantic Annotations for WSDL and XML Schema (SAWSDL), i.e., formalisms which explicitly make use of semantic technologies in different parts of a service description.

One of the primary application areas of SWS is service discovery, which is essentially affected by three steps: (i) The ability of service providers to describe their services, (ii) the ability of requesters to describe their requirements towards services, and (iii) the effectiveness of the service matchmaker, i.e., an algorithm that takes into account a request and finds the best fitting services from a set of service offers.

Service matchmaking that considers semantic information is contemplated by a very agile research community, with a large number of different approaches having been proposed in recent years. A lot of experimentation is conduced concerning the selection of elements from a service description, similarity

metrics, and the combination of the resulting similarity values [142]. State-of-the-art matchmakers are mostly quite inflexible towards differing service domains or need to be adapted manually. This is rather inappropriate as single services as well as service domains might differ to a very large degree regarding basic assumptions towards semantic descriptions of distinct service components or even the availability of a semantic-based domain model. Furthermore, semantic-based matchmaking is mostly based on rather coarse-grained Degrees of Match (DoMs), which allow only a very basic ranking of matchmaking results. As a consequence, it is necessary to complement these values in order to facilitate a sophisticated ranking. If non-logic techniques like *Cosine similarity* or the *Jaccard coefficient*, which are well-known in the field of Information Retrieval (IR) research, are applied, there is always some degree of uncertainty as linguistic-based methods are, for example, unaware of different meanings a term might have [176].

In this thesis, we present matchmakers which are per se solely semantic-based and nevertheless provide an advanced ranking of matchmaking results. In addition, the fine-grained similarity measurement results allow the easy combination with further similarity metrics. Generally, syntax-based similarity measures are only incorporated as a substitute, i.e., if there is no semantic description of a certain service element available. These matchmakers are very flexible due to the incorporation of different (automatic) adaptation mechanisms. Last but not least, matchmaking is not restricted to a certain service abstraction level, but might incorporate information from different parts of a service description.

A second focus of this thesis lies on the ability of service requesters to formulate requirements they wish a service to fulfill. Service registries usually offer a keyword-based query interface, which is not able to incorporate semantic information. In the field of SWS research, service requests are mostly phrased using a "query by example" approach: a service requester needs to model an abstract service description using a service standard like OWL-S or SAWSDL. This approach does not only require a large degree of expertise, but also has other shortcomings, e.g., it is not possible to define a range of values a result set needs to correspond to, or explicitly define which information from distinct service abstraction levels should be regarded in matchmaking. Furthermore, an OWL-S model instance is not sufficient to retrieve SAWSDL-based services and vice versa. Hence, the second major contribution of this thesis will be the development and enhancement of query languages for service discovery. Here, the development of a unified query formalism, which is independent from a certain service registry or service formalism, is the ultimate goal.

In the next section, the contributions in these two fields of research – service matchmaking and service query formulation – will be explained in more detail.

1.1 Goals and Contributions

In this thesis, strategies for the improvement and ease of service discovery based on semantic descriptions are identified, conceived, implemented, and evaluated. Especially, contributions are made in the field of service matchmaking and query formulation for SAWSDL, which is a lightweight semantic extension of WSDL. However, as matchmaking and query formulation aim on an abstract service model (cp. Appendix A.1.3), the developed concepts and technologies are easily transferable to other service formalisms.

Figure 1.1 shows the research topics identified for service-based workflows by the "Service-oriented Computing" research group at the Multimedia Communications Lab at Technische Universität Darmstadt [74, 185]. As can be seen, the contributions of this thesis primarily aim at basic service operations. Regarding Figure 1.1, *service discovery* is addressed by both matchmaking and query formulation, *semantic description* and *service publication* are directly considered in query formulation and utilized in matchmaking. As a vital step in service utilization, discovery influences service composition and particularly *service selection*, too.

The scenario employed in this thesis addresses service discovery, i.e., the process of finding services from a set of service offers based on requirements defined by a service requester. The discovery process is affected by the usefulness of service advertisements and service requests and by the effectiveness of a matchmaking algorithm. Services are usually advertised in a service catalogue called a *service registry* using a certain service description standard like (SA)WSDL or OWL-S. Common service registry standards are, e.g., UDDI (Universal Description, Discovery and Integration) and Electronic Business using Extensible Markup Language (ebXML).

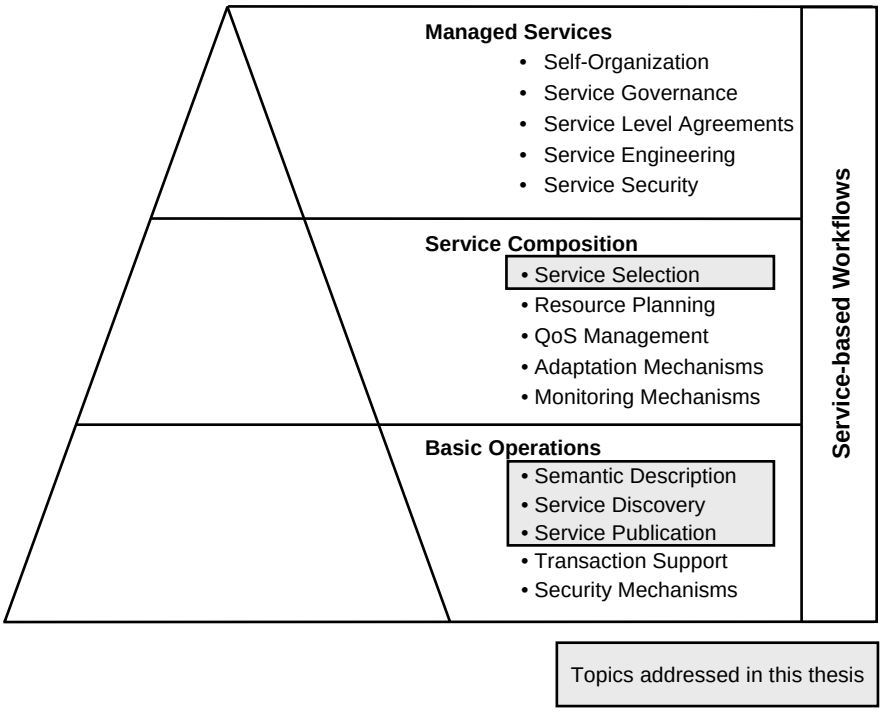


Figure 1.1: Research Agenda for Service-oriented Computing (adapted from [74, 185])

The service discovery scenario can be simplified into the three steps depicted in Figure 1.2:

1. A service provider describes one or more services semantically and/or syntactically and advertises them in a service registry. The actual definition of a service description is not part of this thesis but is assumed to be already available; the construction and representation of the service offer is considered in this thesis in order to facilitate query formalisms for services.

2. Next, a service requester formulates requirements using the query functionalities offered by the service registry. Usually, this search is based on keywords or a "perfect service". In the work at hand, we apply "real" service query languages in a similar way to the query formulation using the Structured Query Language (SQL) in relational databases.

3. As a final step, a matching engine (matchmaking algorithm) processes the query and finds all adequate entities from a set of service offers. The result set to a query is sorted in descending order based on the similarity between query and service offers and returned to the requester; the actual selection and invocation of a service is manually or automatically conducted by the service requester.

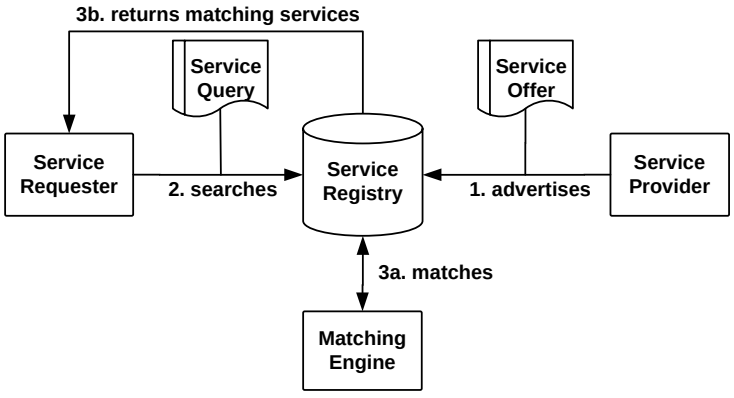


Figure 1.2: Service Discovery Scenario Applied in this Thesis (adapted from [247])

This thesis addresses the functionalities and capabilities of services and disregards other requirements, especially non-functional aspects. This limitation is applied for two reasons: First of all, there is still no commitment on how non-functional requirements should be semantically described, which makes it difficult to develop a universally valid approach to incorporate such information in the service discovery process. Second, the incorporation of non-functional requirements such as Quality of Service (QoS) is a topic on its own and therefore usually considered separately, e.g., in the Ph.D. theses of Berbner [23] and Eckert [74].

The following overview shows the main contributions of the work at hand:

1. Matchmaking of Web services based on semantic information:
   a) Logic-based, self-adaptive SWS matchmaking for SAWSDL with LOG4SWS.KOM: This matchmaker is based on logic-based subsumption matching and is self-adaptive to different basic assumptions regarding the semantic concepts applied in a service description.
   b) Self-adaptive SWS matchmaking for SAWSDL based on semantic relatedness with COV4SWS.KOM: This matchmaker makes use of the notion that the similarity between semantic concepts is based on how specific/generic they are with respect to each other. COV4SWS.KOM is self-adaptive to differing usability of semantic and syntactic service component descriptions on different service abstraction levels. Thus, COV4SWS.KOM can be easily applied to service domains with different degrees of semantic- and syntax-based service descriptions.

   Even though partly relying on well-known principles, LOG4SWS.KOM departs from the rather coarse-grained DoMs usually applied in semantic-based service matchmaking. Instead, a continuous, numerical representation of subsumption DoMs is derived applying a statistical estimator. Syntax-based similarity values are used as a substitute if semantic-based matchmaking cannot be applied.

   Even though the numerical representations of DoMs are automatically derived in LOG4SWS.KOM, there is still some degree of inevitable arbitrariness when defining such values. Hence, we propose the alternative matchmaking approach implemented in the matchmaker *COV4SWS.KOM*, which departs from subsumption matching as foundation for semantic-based matchmaking. Instead, we show that semantic relatedness-based similarity metrics are eligible to compute the similarity between a service request and service offers.

   Both LOG4SWS.KOM and COV4SWS.KOM are (self-)adaptable. However, the adaptability aims on different aspects: LOG4SWS.KOM is adaptive regarding different assumptions made towards semantic concepts applied in a service description. This includes different presumptions regarding what a semantic annotation on a distinct service abstraction level actually denotes as well as the meaning of different subsumption relationships between semantic concepts. In contrast, COV4SWS.KOM is adaptive regarding different degrees of semantic and syntactic description "richness" on distinct service abstraction levels. This addresses the fact that the element description on a bottom level of a service description might be very rich while on a higher abstraction level there is perhaps no, however specified, description available (and vice versa).

2. Query Languages for semantic Web services:
   a) Integrating semantic discovery in Universal Description, Discovery and Integration (UDDI) using the SPARQL Protocol and RDF Query Language (SPARQL): This includes the insertion of SAWSDL-based Web services in UDDI, the creation of a SPARQL-aware service request interface, and the involvement of LOG4SWS.KOM in UDDI query processing.
   b) A unified querying formalism for SWS: The former approach makes use of already existing standards and technologies and is therefore easily integrable into existing service frameworks. However, the reuse of existing standards imposes some constraints, as especially SPARQL has not been initially designed for query formulation for SWS, and lacks the amenities which ease the formulation of service queries. Thus, the second approach presents a more advanced query formalism for SWS by enhancing a common query standard, namely SPARQL, in a way convenient for SWS retrieval. Through the application of abstract service data and query models, this approach is standard-independent and therefore, easily transferable to further

SWS and registry standards. As a proof of concept, it is shown how the unified querying formalism is integrated in ebXML, providing the possibility to integrate different matchmakers and SWS formalisms.

Both approaches are based on an extensive analysis of requirements towards a unified service query formalism. The definition of service requests is addressed by the deployment and development of two related query languages, namely a slightly extended SPARQL-syntax and the SPARQL-based SWS Structured Query Language (SWS2QL). The aim is to progress from the query formulation usually applied in (semantic) Web service retrieval to a more convenient format. This research aims in two directions: With SPARQL, a format is adopted which is widely accepted as the de facto query language for the Semantic Web and is of great importance to this research community [26]. By extending SPARQL in SWS2QL, a service requester is allowed to address different service abstraction levels, incorporate and parameterize matchmakers, and define thresholds, *without* knowledge about the underlying service registry or SWS description formalism. This is achieved through the deployment of abstract service data and query models. This way, a unified, transferable service query formalism is provided.

Furthermore, ebXML is a primarily commercially applied service registry standard – in the opinion of the author, the integration of semantic service matching into such a standard is crucial for advancing the acceptance of SWS in the software industry. Hence, a general framework for the integration of SWS descriptions and different service matchmakers into this registry standard, is also presented.

1.2 Outline

The remainder of this thesis is structured as follows: Chapter 2 describes the conceptual and technical background necessary to understand the content of this thesis. To start with, SOC is explained with special regard to its technical aspects and possible actors involved. Afterwards, the terms *service* and *Web service* are defined and the most important Web service standards and technologies are presented. Furthermore, the incorporation of semantics in service descriptions is examined. This starts with a description of the issues that are tackled by augmenting service standards with explicit semantic information. Afterwards, a brief introduction to semantics for Web services and ontologies (which constitute the most common choice to define semantic concepts for SWS) is provided. Third, an overview of SAWSDL and OWL-S as the two SWS standards that are employed in this thesis is given. The last part of Chapter 2 covers service discovery and provides an introduction to the two topics regarded in this thesis, i.e., *service matchmaking* and *query formulation*.

While we present the latter in more detail in Chapter 4, the former is addressed in Chapter 3. To start with, shortcomings of current matchmaking approaches are identified. The first matchmaker presented is LOG4SWS.KOM (Section 3.2), a logic-based matchmaker relying on Description Logics (DL) and subsumption matching. Here, requirements towards LOG4SWS.KOM that arise from the previously defined shortcomings are identified. Afterwards, the matchmaking approach is explained regarding three aspects: (i) selection of matching levels, (ii) choice of similarity measures, and (iii) selection of a matching algorithm. In addition, a non-logic-based fallback strategy, which can be applied if semantic information is missing for distinct elements of a service description, is presented. Subsequently, the implementation of LOG4SWS.KOM is covered. The second matchmaker – COV4SWS.KOM – aims at the adaptation to varying quality and usability of information on different service abstraction levels, and therefore, in different service domains. Instead of logic-based similarity metrics, approaches from the field of semantic relatedness in ontologies are applied. Again, the underlying ideas and methodologies are explained, followed by the matchmaker's implementation. Afterwards, both matchmakers are extensively evaluated. The evaluation of both matchmakers is conducted using a well-established test data collection of SAWSDL-based service descriptions and with regard to IR metrics like precision and recall. Finally, the quantitative evaluation results as well as the qualitative aspects of the approaches are discussed. The chapter ends with a discussion of the related work and a recapitulation of the findings.

Chapter 4 addresses the development of query formalisms for SWS. The aim is to allow an easy definition of requirements a requester has towards services. First, the problems of commonly applied query formats are discussed and requirements towards query languages for SWS are defined. Afterwards, these

requirements are addressed in two different approaches. The first solution, which is implemented for the UDDI registry standard, is based on well-established standards, i.e., SAWSDL and SPARQL. However, adherence to these existing technologies (especially SPARQL) leads to some shortcomings, which are regarded in the second solution. Instead of simply using SPARQL, the second approach enhances SPARQL by special features for the retrieval of SWS. Therefore, a unified query model for SWS is developed, which is the foundation for the SPARQL extension. The thereupon developed query language – SWS2QL – is integrated into ebXML. Furthermore, common ways to incorporate SWS formalisms into this registry standard and to access matchmaking facilities for SWS from within ebXML are presented. The chapter ends with an overview of the related work and a summary of the findings.

This thesis closes with a summary of its results and gives an outlook on possible future work in the examined field of research.

2 Background

This thesis addresses the problem of Web service discovery based on semantic information. Hence, knowledge and technologies from SOC and Web services in general, semantic Web services, and service discovery are needed. In order to keep this chapter relatively brief, technical specifications have been relocated to Appendix A.

The first section covers SOC in general. This includes selected definitions as well as an introduction to Web service and service registry standards and formalisms. Afterwards, major aspects of semantic Web services will be presented in Section 2.2. The concluding section covers service discovery with special regard to the incorporation of semantic information in query formulation and matchmaking.

2.1 Service-oriented Computing

According to the IEEE, the field of Service-oriented Computing ranges from IT topics like service principles and service technologies to business-oriented aspects like services consulting and delivery or services solutioning and management [259]. This broad spectrum is also recognizable if regarding the SOA paradigm: Originally, the term "SOA" has been coined in 1996 by Gartner and addresses "a style of multitier computing that helps organizations share logic and data among multiple applications and usage modes" [224]. Since 1996, several publications have redefined this term. Some definitions emphasize the technical aspects of SOA (e.g., [116, 148, 171]) while others combine business and technical aspects (e.g., [76, 204]).

Depending on which definition of SOA or SOC is applied, a different view on the term "service" may be deployed. This can be traced back to the fact that SOC research has been influenced by three different research communities: business science, information science, and computer science [15]. If taking primarily the business aspects of SOC into consideration, a service might be considered as one particular business functionality or process step [76]. However, in this thesis, another point of view is adopted, as the presented solutions primarily aim on technical aspects of services while business-related aspects of services are only of secondary importance.

Accordingly, we make use of a definition provided by Preist and Baida et al. The authors separate between the actual provision of something of value, which is called a "service", and "Web services", which are actually a specific application providing a WSDL and SOAP interface [15, 214]. Baida et al. explains the differences as follows: As the term "service" is primarily used by business scientists, it describes the business perspective of a service. "Web services" are primarily considered by computer scientists, hence, this term describes the view of this research community. Furthermore, Baida et al. also make use of the term "e-service" in order to describe services from an information science perspective. However, we drop this term in this thesis, as it is not very common and the information science perspective, i.e., the business process view as defined by Baida et al. [15], is not of primary interest. In the following, we differentiate between services and Web services as proposed by these authors. However, as (semantic) Web services are in the focus of the research at hand, the terms "service" and "Web service" will be used synonymously unless indicated otherwise. This applies to all composed terms, too, i.e., "service discovery" and "Web service discovery" address the same task.

A basic description of the interactions in SOC is the *Publish-Find-Bind-Execute* model depicted in Figure 2.1. Services are published, found, and bound/executed in a model made up from three different roles [88]:

A *Service Provider* offers one or more services, i.e., defined software components which possess a service interface. A provider can publish services in a *service registry* in order to make it available to a larger user community. Such a registry is often hosted by a service broker.

A *Service Consumer* (also *Service Requester*) requests services. If the service, which will be invoked, is known beforehand, the service consumer directly binds the service. If this is not the case, a service broker needs to be involved in the invocation process. After a service has been found, the service is bound and invoked/executed.

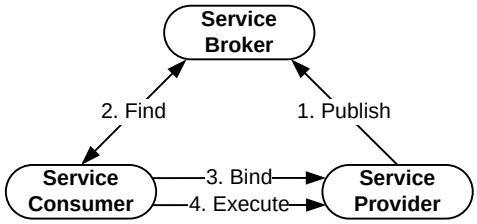


Figure 2.1: Publish-Find-Bind-Execute Model (adapted from [88, 163])

A *Service Broker* could be comprised for many reasons. In most scenarios, the broker provides a service registry where services can be published and requested.

A provider is not necessarily a human being or an organization but might also be a software agent [242, 243]. As a service provider can also be a service consumer and vice versa, the parties could be different peers in a Peer-to-Peer (P2P) network [237].

The ultimate goal of invoking services from brokers or registries on the Web is the ad-hoc collaboration between partners, i.e., the dynamic binding of Web services during the runtime of a workflow or of single services [8, 25]. If applying automatic composition of services and workflows in different domains, it is necessary to have many services at hand. Papazoglou suggests the concept of service markets, where services could be offered and requested [203]. Such service markets act as service brokers which offer different intermediary services to both service consumers and providers. Following the vision of the "Internet of Services", services for all areas of life and business will be available on the Web [197]. Of course, this makes it necessary to provide a meaningful service description and means to invoke services using the Internet. Here, Web service standards and formalisms like, e.g., WSDL, SOAP, and UDDI come into play.

The core component of a service market is a service registry or service repository. While the aforementioned entity is some kind of a service catalogue, i.e., a place where service providers can advertise their services and information about their bindings, the services are actually hosted by the service providers themselves. In contrast, a service repository is not only a service catalogue but hosts the advertised services, too (cp. Section 2.1.3). Apart from the retrieval of a service, a service market respectively service broker can offer additional functionalities, e.g., for accounting, the monitoring of service invocations, or rating of service providers [24, 218].

Especially in large scenarios, the manual discovery of services is cumbersome and might lead to suboptimal results, as a human being is not able to cope with a very large number of services. Hence, in such scenarios, it is necessary to identify services which offer the right functionality, with as little human intervention as possible. But even in smaller scenarios, an automated service discovery will help to facilitate the actual service invocation. In the best case, service discovery and selection can be carried out automatically. Here, the description of a service plays an important role.

For service discovery, the description of a Web service as well as its presentation in a service registry are of primary interest. In the following, Web services will be introduced using a technology focus (Section 2.1.1), with special regard being paid to WSDL (Section 2.1.2). Afterwards, service registries will be introduced (Section 2.1.3). In this thesis, two service registry standards will be deployed, namely UDDI and ebXML – these standards will be further presented in Appendix A.2.

2.1.1 Web Services

To concretize the aforementioned definition of a service, we make use of a definition of the W3C. Accordingly, a Web service is defined as a

> "software system identified by a URI, whose public interfaces and bindings are defined and described using XML. Its definition can be discovered by other software systems. These systems may then interact with the Web service in a manner prescribed by its definition, using XML based messages conveyed by Internet protocols." [10]

Figure 2.2 shows a minimalist infrastructure for the usage of Web services by extending Figure 2.1 with the three core Web technologies WSDL [32], UDDI [62], and SOAP [188]. As it can be seen, UDDI is

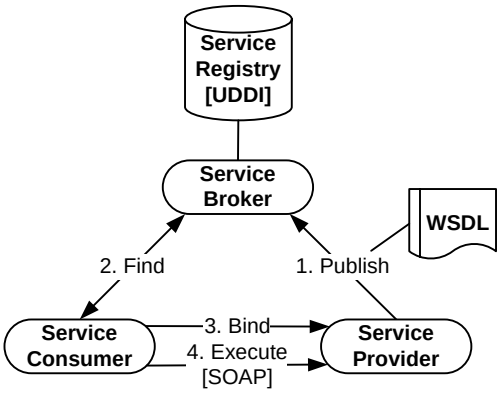


Figure 2.2: Web Service Technologies

used as service registry technology in order to provide an infrastructure where services can be published and retrieved. As such, the UDDI registry is hosted by the service broker. WSDL is used in order to describe Web services; the service advertisements in the UDDI registry are also based on WSDL. SOAP is used in order to invoke a service and exchange messages between applications. Combined, WSDL, UDDI, and SOAP facilitate the application of service-oriented concepts on the Web, e.g., to publish, find, bind, and execute services [205]. Other specifications, e.g., for describing Web service policies [217, 223] are typically built upon these standards [8].

As WSDL and SOAP are kept rather simple, it is necessary to incorporate further Web service standards in order to provide a more sophisticated service description or to address QoS aspects. Figure 2.3 shows an overview how different standards augment the basic Web service technologies mentioned above. These standards are arranged on five layers which compose the *Web Services Standards Stack* [33, 88, 210].

Service Composition is realized through a language to describe a service flow. Nowadays, the Web Services Business Process Execution Language (WS-BPEL) is the standard language to describe business processes or workflows made up from services. WS-BPEL can also be used in order to describe a service flow which is not addressing a business process [9].

Description of Web services is usually done in WSDL. However, some parts of the Web service are described in the XML Schema Definition Language (XSD) – with regard to WSDL 2.0, types of inputs and outputs are usually described in XSD [32]. As we will see in the following, WSDL is made up from basic components describing a service's interfaces, operations etc. Further aspects like, e.g., policies are not regarded and have to be included in the service description by making use of one of the numerous *WS-*$*$ standards [204]. In Figure 2.3, *WS-Policy* is depicted as one example for a standard which enhances the WSDL-based service description [249].

Quality of Service aspects are also not explicitly regarded in SOAP and WSDL. As a result, the description of QoS parameters is realized using further WS-$*$ standards. In Figure 2.3, examples for the realization of transactions (*WS-Transaction* [80, 84, 168]), reliable messaging (*WS-ReliableMessaging* [66]), and security (*WS-Security* [190]) are depicted. While WS-Policy is augmenting WSDL-based service descriptions, WS-Transaction, WS-ReliableMessaging, and WS-Security are extensions for SOAP.

Interaction between and with Web services is based on message exchange. While SOAP is the most important standard, other interaction protocols are also possible [56]. Here, the Representational State Transfer (REST) paradigm should be highlighted, as *RESTful* Web services have gained considerable importance in recent years [220].

Network describes the possible protocols that can be applied in order to actually send SOAP messages. Here, HTTP(S) is the most common protocol for both SOAP and REST [96, 220].

In the context of this thesis, WSDL and UDDI respectively ebXML are the most important Web service standards. In the following section, WSDL will be briefly introduced. Afterwards, service registries will be regarded. As SOAP is not in the focus of this thesis, we refer to the SOAP 1.2 specification for further details [95, 96, 188].

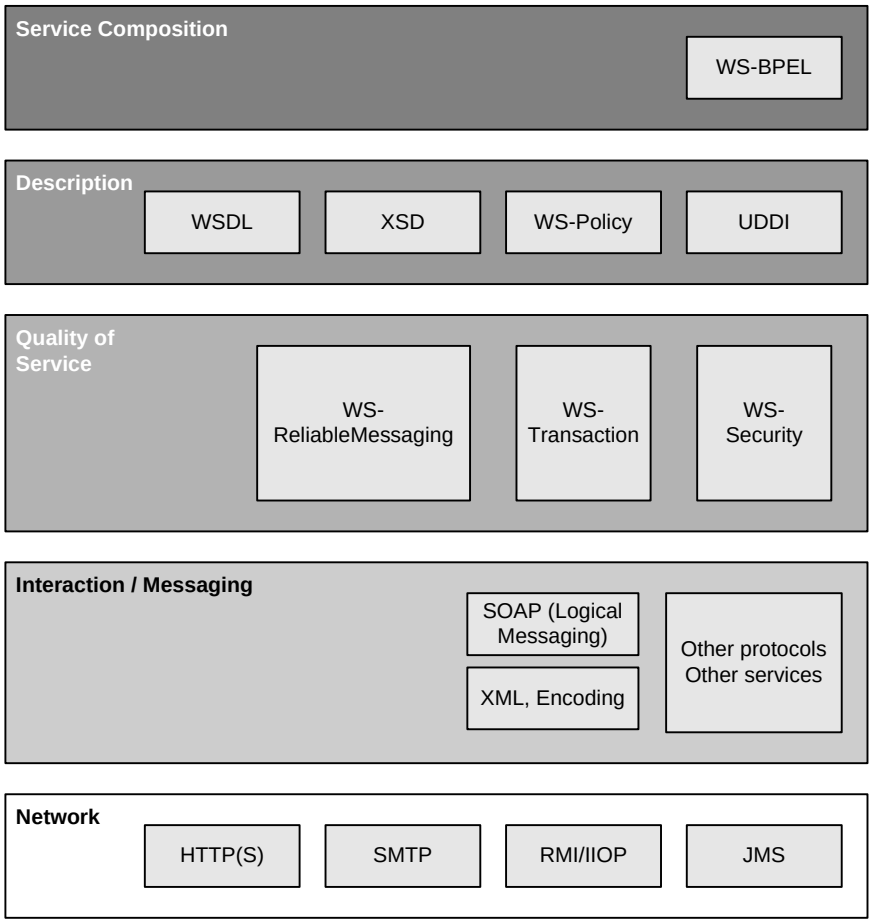


Figure 2.3: Web Services Standards Stack (adapted from [33, 88, 210])

2.1.2 Web Service Description Language

As the name implies, WSDL is a description language for Web services, i.e., a formalism, which represents a Web service on an abstract level as well as its concrete definition (cp. Figure 2.4). At the time of writing, WSDL 2.0 is the current recommendation by W3C and is intended to replace its predecessor WSDL 1.1 in the near feature [32]. Hence, in this thesis, we will make use of WSDL 2.0. In the following, "WSDL" always refers to WSDL 2.0 except if indicated otherwise. However, unless explicitly defined otherwise, it is possible to transfer all concepts and results developed in this thesis to WSDL 1.1 based on corresponding mappings respectively adaptations of the algorithms (cp. Appendix A.1.3).

WSDL is XML-based and independent of the underlying programming language of the service implementation and the platform the service is running on – it encapsulates the service functionalities so that a service consumer does not have to handle different technological infrastructures at the service provider's side. The abstract and concrete parts of the service definition in WSDL 2.0 are wrapped in the root element *description*. While the abstract part includes the service interfaces, corresponding operations, input and output messages, the concrete part is made up from the service binding, endpoint, and actual service [32]. The abstract part of a WSDL document advertises *what* a service does while the concrete part defines *how* a service can be consumed and *where* it is located. Thus, a WSDL document defines a contract between service consumer and provider with regard to functional and selected non-functional aspects of the service [204].

In the context of service discovery, the description of *what* a service does is of primary interest. Hence, in the following, the abstract part of a WSDL-based service description – interfaces, operations, and messages – will be analyzed in more detail. These service components constitute the *service abstraction levels* of WSDL: Functionalities, i.e., interactions between a client and a service, are described by abstract *operations* [57]. A set of operations defines a service *interface*. For each operation, a sequence of *messages* a service is able to send and/or receive may be defined.

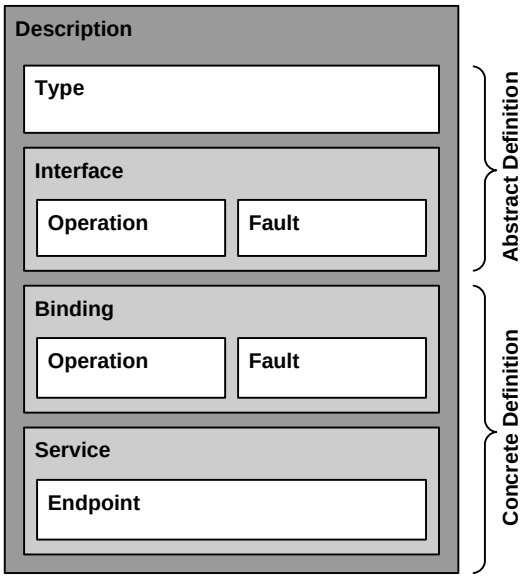


Figure 2.4: WSDL 2.0 Conceptual Model

In WSDL, input and output messages are defined using *types*. Even though message types could be defined in different schema languages, XSD has been established as the de facto standard schema language for message types, as it is natively supported by WSDL 2.0 [32]. Although the WSDL 2.0 specification makes it necessary to declare message types as single elements at the topmost level, each element may have a substructure. Thus, it is possible to define simple and complex message types [32]. Elements from these message types are often called *parameters*. Together, inputs and outputs form the so-called *service signature*.

Per se, WSDL is purely syntax-based, i.e., service functionalities, message types etc. are only defined on a syntactic level. However, there have been some attempts to enrich WSDL with semantic annotations, e.g., using SAWSDL [79]. This will be further examined in Section 2.2.

2.1.3 Service Registries

Service registries or repositories are essentially service catalogues which are a core component of a Web service execution environment like WSQoSX or METEOR-S [24, 198]. As it is the case with the terms "service" and "Web service" (see above), "registry" and "repository" are often used interchangeably to refer to service catalogues in general, although each term exhibits a distinct view on service management: While service *registries* represent the technical perspective, service *repositories* represent the service management perspective, i.e., a more business-based view [116].

In the real world, (business) services are described and listed in yellow page directories, so that customers are able to search for suitable services and their providers. Concerning software-based services, this requirement also needs to be satisfied. For this, service registries are applied, which serve as a directory for service advertisements and offer discovery functionalities. Besides the syntactic description of service advertisements, further information like, e.g., service classification using taxonomies or non-functional details may also be stored in a service registry.

While service *registries* only provide directory services and references to artifacts, i.e., metadata, the actual artifacts are stored in a service *repository* [116]. The latter are used to manage service interfaces, contracts and additional service information concerning usage fees, available service levels, information about the service provider, or technical and security constraints [116, 148]. Registries and repositories may either be provided as distinct entities or in a single system. In the first case, the repository should be referenced from the registry in order to be able to retrieve the details of a service [116]. Typically, proprietary databases are used to establish a registry/repository, in which a (formal) service contract is registered/stored for each service together with some administrative data [148]. In general, a service

registry offers an interface to register, update and discover services. A registry can either be made available for internal use or to the general public. In the latter case, multiple service providers may offer similar services in order to establish a marketplace for services [116]. Within this thesis, the abovementioned functionalities of service registries are of primary interest. Hence, even if regarding a standard like ebXML, which incorporates both a registry and a repository, in this thesis, we will refer to the registry component. Thus, the terms "registry" and "repository" are used interchangeably.

UDDI is without question the most-applied service registry standard which has been used for the publication and retrieval of SWS from the very beginning of SWS research. An overview of approaches to augment UDDI with different SWS formalisms can be found in Section 4.5. However, with proprietary solutions and ebXML Registry, there are adequate alternatives to UDDI. In this thesis, both UDDI and ebXML Registry will be applied as repositories for Web services in Chapter 4. A more detailed introduction to these standards can be found in Appendix A.2; a discussion on their respective pros and cons can be found in Section 4.3.6.

2.2 Semantic Web Services

Technologies like WSDL, UDDI, SOAP, or WS-BPEL are primarily motivated by the syntactic interoperability of software components over the Web [21, 258]. They rely on the syntactic description of Web services' functionalities and properties in XML. Thus, common standards lack of machine-interpretable information regarding functional and non-functional aspects.

Intuitively, it could be assumed that a service's purpose can be derived from the description of its inputs and outputs. However, this is not necessarily the case, because, e.g., the WSDL standard does not impose any conventions regarding the naming of elements. WSDL is XML-based; as XML documents cannot be necessarily interpreted without human intervention, this is also the case for WSDL. Simply speaking, in contrast to a human, a computer does not understand the underlying semantic meaning of terms. With regard to Web services, this leads to the point that Web services, which offer similar functionalities, could possess substantially differing interfaces and operations. The other way around, services which possess the same interfaces could offer completely different functionalities [87, 236]. This is a major shortcoming which especially affects service discovery.

Paolucci et al. state that this problem should be addressed by rising "above [...] superficial differences in the representation of interfaces of services and to identify the semantic similarities between them" [201], i.e., a service description should not be only syntax-based but should also be augmented by semantic concepts describing single service components. Here, ideas and technologies of the *Semantic Web* come into play: At the beginning of the millennium, Tim Berners-Lee proposed the idea of the Semantic Web [27]. He argued that huge benefits could arise from the information and functionalities available on the Internet if it were interpretable by machines. Accordingly, Berners-Lee proposed to extend the Web with semantics in order to give information a well-defined meaning and provide a layer of machine-interpretable data [27, 226, 240]. In particular, Berners-Lee already considered software agents which could work together in order to collect and process (semantic) data. He also defined that in a "process, called service discovery", it would be necessary to describe an agent's functionalities semantically by "a common language to describe a service in a way that lets other agents 'understand' both the function offered and how to take advantage of it" [27].

As of today, several technologies like the Resource Description Framework (RDF), or Web Ontology Language (OWL) have been exploited respectively standardized and provide the basic functionalities of the Semantic Web. Figure 2.5 shows the different aspects of the Semantic Web [26]. The Semantic Web is in the focus of numerous research projects and semantic technologies are applied to various fields like knowledge management, business intelligence, Web 2.0, and finally, SOC and Web services [22]. The latter is quite obvious, as Web service standards are based on XML, which is also one of the building blocks of the Semantic Web. This makes it possible to easily combine Web service and Semantic Web technologies to create semantic Web services which are defined as Web services whose "properties, capabilities, interfaces, and effects are encoded in an unambiguous, and machine-interpretable form" [183].

In one of the seminal papers on SWS, McIlraith et al. proposed the usage of Semantic Web technologies in order to markup Web services to make them machine-interpretable and accordingly facilitate automatic Web service discovery, execution, composition, and interoperability [183]. Likewise, Tsetsos et al. empha-

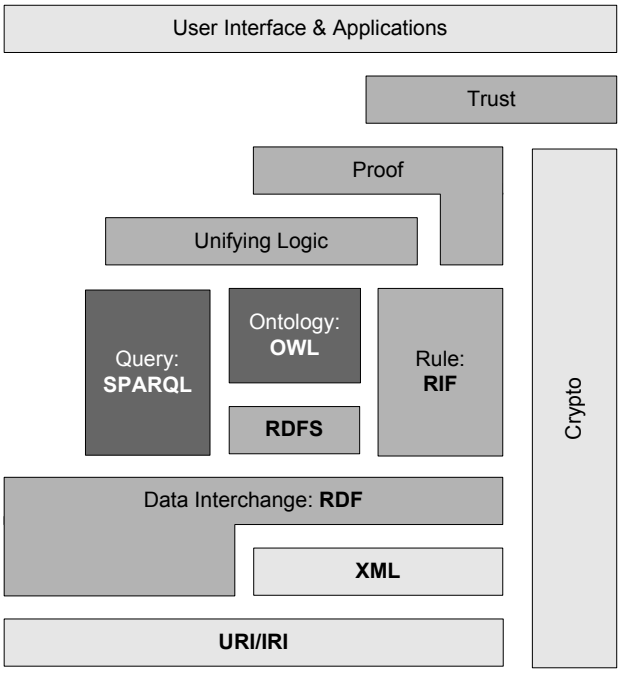


Figure 2.5: Semantic Web Reference Architecture Version 4 (taken from `http://www.w3.org/2001/sw/`)

size the improvement of Web service representation expressiveness and logic-based reasoning as reasons for the demand for semantics in Web services [247].

Regarding Web service research in general, the integration of semantic technologies is often coined as one of the grand challenges that need to be met in order to facilitate the success of Web services on a broad scale (cp., e.g., [30, 149, 179, 180, 206, 207]). Amongst others, adding semantics to Web service standards is deemed as a major success factor in order to facilitate service foundations, service management, and service engineering [44, 208]. As it was mentioned before, Web services are commonly considered as part of business processes or workflows. Hence, the usage of SWS is also one of the key factors in order to facilitate the automatic composition of workflows and business processes from Web services [47, 105, 184], for example by adding support for SWS to WS-BPEL [175, 195]. In fact, semantics can be added to numerous stages of the Web Services Standards Stack presented in Figure 2.3, thus constituting a vertical layer that augments the horizontal layers of the original stack [43].

In the following, the most important concepts and technologies regarding SWS will be briefly introduced, namely semantics in general (Section 2.2.1), ontologies (Section 2.2.2), and the SWS formalisms SAWSDL and OWL-S (Section 2.2.3).

2.2.1 Semantics for Web Services

Kashyap and Sheth define "semantics of information (...as ...) the meaning and use of information" [117]. In contrast "syntax" defines the structure of data, e.g., the WSDL syntax defines how a WSDL document has to be structured, which elements are allowed etc.

Generally, Semantic Web content can be divided into data and metadata [118]. While the data represents the actual content, metadata can be defined as "data or information about data" [118]. In general, the application of metadata descriptions is twofold: On the one hand, an abstraction of the representational details can be achieved (e.g., format, organization) while capturing the information content, and on the other hand, domain knowledge can be associated with the data allowing to make inferences such as the relevance of the data or relationships to other pieces of information. The key to achieve machine-processability of content is the grounding of terms used in metadata descriptions in well-defined, standardized vocabularies [118]. For this, *ontologies* are a common format.

Regarding Web services, semantics can be used to give certain service components a *meaning*, e.g., to provide detailed information what an operation, interface or message from a WSDL description provides. Apart from enhancing already existing Web service description components by meanings, the integration

of semantics into Web service standards has also resulted in the definition of new elements, which have not been regarded in non-semantic Web service standards. *Preconditions* and *effects* (sometimes also referred to as *postconditions*) are well-known examples for such new constructs. Preconditions and effects might be logical conditions that need to be fulfilled before the service can be carried out respectively describe changes in the world after the service has been executed [231]. In contrast to inputs and outputs, which are stateless, preconditions and effects are giving information about a state before respectively after a service has been invoked. Together with inputs and outputs, preconditions and effects form the *service profile* [129]. Preconditions and effects have been proposed in WSDL-S [6] and OWL-S [178] but have not been explicitly incorporated in SAWSDL [79]. However, Vitvar et al. have shown how preconditions and effects could be integrated into SAWSDL [251].

Sheth et al. distinguish four different kinds of semantics for Web services [87, 209]:

Data semantics formally define data in input and output messages of Web services.

Functional semantics formally define the capabilities of Web services, i.e., by defining preconditions and effects and semantically annotating interfaces and operations.

Non-functional semantics reference QoS and general policy requirements/constraints.

Execution semantics describe the execution of services and operations.

In order to define the meaning of distinct service components by semantic annotations or enhancements of a service description, it is necessary to have a domain model which can be used as a *knowledge base*. Most probably, the best-known knowledge base format are *ontologies*. However, there are other formats, too: Cardoso defines four levels of semantics with respect to their expressiveness: While *controlled vocabularies* only define a list of terms with an unambiguous definition, *taxonomies* also arrange these terms into a hierarchy. A *thesaurus* allows to relate terms from a taxonomy by defining equivalences, homographs, hierarchical relationships, and associations. *Ontologies* provide the highest expressiveness of these four representations by also providing richer semantic relationships between terms and attributes [46]. In line, Berners-Lee et al. state [27]:

> "The most typical kind of ontology for the Web has a taxonomy and a set of inference rules. The taxonomy defines classes of objects and relations among them."

As SWS standards heavily rely on ontologies, the next section will present these structures in more detail.

2.2.2 Ontologies

In the field of Artificial Intelligence (AI), the usage of ontologies has got a long tradition as means to describe the knowledge about a particular domain [52]. In Semantic Web research, ontologies provide the foundation for machine-processable data and allow to exchange information between people and machines by both syntactic and semantic means [172]. Ontologies can address several domains and be described in many different forms and serializations – as a result, there is no commonly agreed definition of the term "ontology", and available definitions differ across scientific communities [158]. However, the following quote by Gruber is widely accepted as a common definition of an ontology from a more technical view [94]:

> "An ontology is a formal, explicit specification of a shared conceptualisation."

This definition highlights distinct features an ontology needs to address: First of all, it is *formally* specified, i.e., an ontology makes use of a defined ontology language. Second, *conceptualisation* refers to an abstraction of a domain which includes the relevant concepts in that domain. Third, an ontology is based on *shared* knowledge, i.e., it represents an agreed viewpoint.

A more comprehensive explanation of the intended purpose of an ontology in computer science is provided by Lacy [158]:

> "Computer science ontologies serve a similar function as database schemas by providing machine-processable semantics of information sources through collections of terms and relationships. The semantics support a shared and common understanding of a domain that can be communicated between people and software."

Differently stated, according to Lacy, ontologies define the meaning of terms with respect to a specific domain of knowledge, and how these terms are related to each other. For example, with respect to the domain of English language, the terms “yes” and “no” may express agreement and disagreement respectively. Accordingly, they are *antonyms*, because they mutually express the opposite – this could be represented in an ontology by a corresponding relationship between these terms.

In general, relationships between semantic concepts are defined by inference rules. The rules allow to deduct relations between classes that have not been explicitly stated. For instance, let us consider the above example with the English language ontology, containing “yes” and “no”, again. Assume that the ontology also states that “yeah” is a synonym of “yes”, i.e., their meaning is equivalent. Given a corresponding inference rule, we could implicitly reason that “yeah” is an antonym of “no”, as “yes”, which is a synonym of “yeah”, is an antonym of “no”. As this simple example demonstrates, inference rules provide a powerful extension to taxonomies, allowing to derive knowledge that has not been explicitly stated.

In conjunction with the Semantic Web activities of the W3C, two major standards have arisen for the representation of ontologies, namely RDF Schema (RDFS) [36] and OWL [182]. RDFS and OWL are based on RDF, an XML-based standard which allows to define *triples*. A triple defines a relationship between an arbitrary *subject* and an *object* using a so-called *predicate*. In RDF and OWL, subjects, objects, and predicates are expressed through Uniform Resource Identifiers (URIs) respectively Internationalized Resource Identifiers (IRIs) (IRIs are a generalization of URIs, allowing to make use of general Unicode characters). Thus, RDF provides a very generic and flexible framework for knowledge representation. A more detailed discussion of RDF, RDFS, and OWL can be found in Appendix A.3.

2.2.3 Semantic Web Service Formalisms

Since 2001, several initiatives have contributed to SWS standardization with the DARPA Agent Markup Language for Services (DAML-S) and its successor Web Ontology Language for Web Services (OWL-S), the Web Service Modeling Ontology (WSMO) and Web Service Modeling Language (WSML), and Web Service Semantics (WSDL-S) respectively Semantic Annotations for WSDL and XML Schema (SAWSDL) being the most popular outcomes of these efforts [79, 81, 178]. A whole research community has been established which addresses the (semi-)automatic annotation, retrieval, and composition of Web services based on Semantic Web concepts as well as other SWS aspects [179, 180].

Today, SAWSDL is the only official standard for SWS recommended by the W3C, even though it provides quite a lightweight approach to semantic descriptions for Web services. In contrast, OWL-S and WSML provide much richer semantic formalisms for Web services. In this thesis, SAWSDL will be in the focus, while OWL-S will also be regarded in order to show how the approaches developed can be applied to more heavyweight Web service semantics. However, the query formalisms and matchmaking algorithms presented in Chapters 3 and 4 are not directly applied to SAWSDL and OWL-S, as their deserialization and parsing takes a considerable amount of time. Instead, SAWSDL and OWL-S formalisms are mapped to the Abstract Truncated Service Model (ATSM) presented in Appendix A.1.3.

Semantic Annotations for WSDL and XML Schema

SAWSDL constitutes a lightweight extension to WSDL, which introduces three new attributes that can be used in conjunction with the already existing WSDL 2.0 components presented in Section 2.1.2 [4, 79]. These SAWSDL attributes are:

modelReference which points to one or more URIs of arbitrary concepts that semantically describe a WSDL component.

liftingSchemaMapping which points to one or more URIs that allow for the lifting of data from XML to a semantic level.

loweringSchemaMapping which points to one or more URIs that allow for the lowering of data from the semantic level to XML.

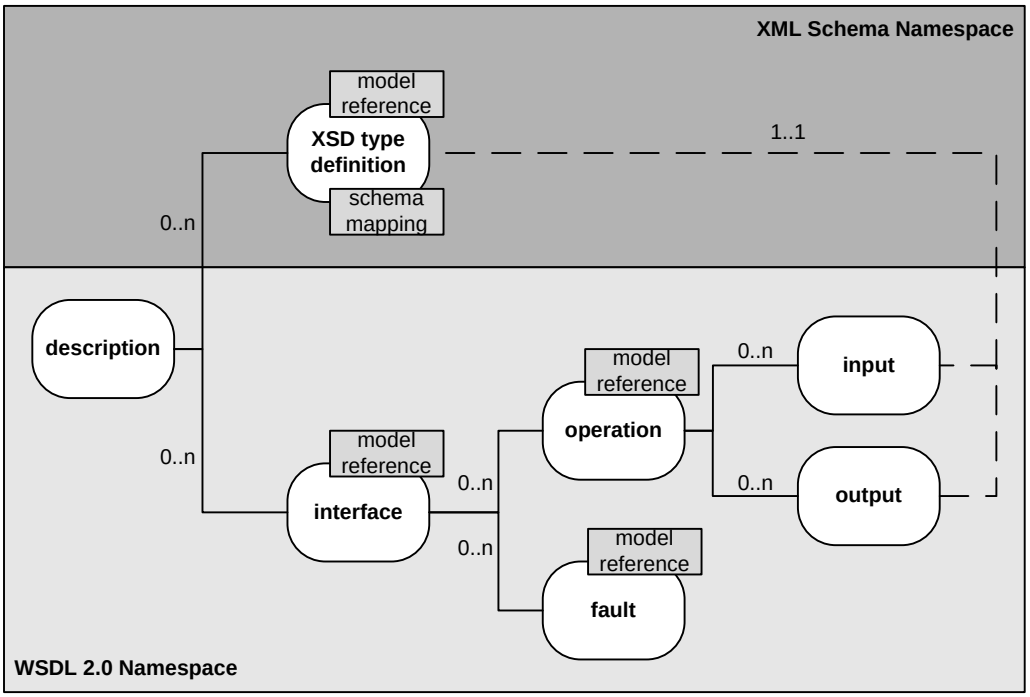


Figure 2.6: Composition of a WSDL Document with SAWSDL Extensions (adapted from [132])

An overview of the WSDL components, along with the SAWSDL extensions, is provided in Figure 2.6. A more detailed example (SA)WSDL document can be found in Appendix A.1.1. As it can be easily seen, SAWSDL augments the service description by pointing to semantic concepts. These semantic annotations allow deriving a service's purpose, or automatically conducting comparisons between a pair of services.

SAWSDL does not impose any restrictions on what a semantic annotation eventually means nor the type of semantic concept that is addressed, i.e., it does not provide any specific semantics by itself [144]. Regarding the former, there is no definition, what, i.e., a model reference on interface level means. However, the SAWSDL specification states that on interface level, a model reference might be a categorization, while on operation level, a model reference might specify a high level description of the operation – both applies to functional semantics. On message schema respectively types level, model references specify data semantics as defined by Sheth et al. (cp. Section 2.2.1) [87, 209]. This "meaning" of semantic annotations is compliant with the classification made for *WSMO-Lite* by Vitvar et al. [251] and is also applied in this thesis.

Regarding the type of semantic concepts, SAWSDL does also not impose any restrictions as long as a semantic concept is identifiable by a URI [79, 144]. This is an advantage as it allows for a maximum of flexibility in annotating functional service descriptions. Yet, in several application scenarios, this fact poses a problem. Generally, this is the case if the concepts need to be processed and interpreted in some form. In this thesis, we will thus assume the semantic concepts to be formally defined in an OWL DL-based ontology (cp. Appendix A.3.2). This restriction is a common assumption in conjunction with the use of SAWSDL, especially in matchmaking.

Web Ontology Language for Web Services

An alternative to SAWSDL is the Web Ontology Language for Web Services (OWL-S), which is a W3C member submission from the year 2004 [178]. Compared to SAWSDL, OWL-S is a heavyweight approach that defines a full-featured, OWL-based ontology for the semantic description of services [7]. In line with SAWSDL, OWL-S may also be regarded as a complement to WSDL, not as a substitute. Whereas SAWSDL defines a bottom-up mechanism for enriching functional descriptions with semantic annotations, OWL-S follows the opposite principle, i.e., a top-down approach. It describes the semantic characteristics of a service, which may then be grounded in (i.e., mapped to) a functional description.

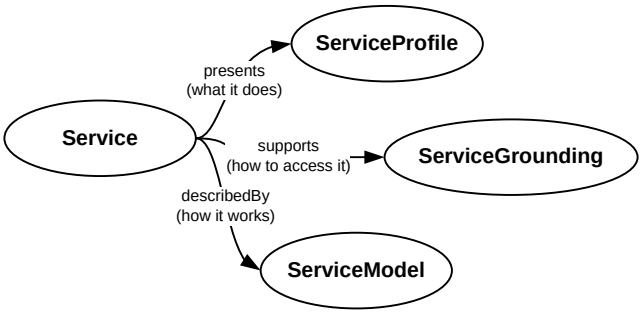


Figure 2.7: Top Levels of the OWL-S Service Ontology (taken from [181])

The goal of OWL-S is to support "automatic Web service discovery, invocation, composition and interoperation" [181]. An overview of the OWL-S service ontology is depicted in Figure 2.7. In detail, the components serve the following purposes:

The *service profile* provides a description of the service, which can be used in conjunction with service publication and discovery. The profile contains information that is required at the time of service publication at a registry and subsequently allows to discover the service, e.g., about functional parameters as inputs, outputs, preconditions, and effects, as well as about non-functional parameters as a, e.g., service category or quality rating.

Following discovery, the *service model* contains information, which is required to control the interaction with the service. Namely, it describes stateless atomic processes and more complex composite processes, which maintain a state and may be controlled in their execution flow through control structures. Depending on whether processes produce information or changes in the state of the world, they are associated with inputs and outputs or preconditions and effects. Whereas inputs and outputs are to be described in a DL-based language (cp. Appendix A.3.3), such as OWL DL, preconditions and effects are expressed in logic formalisms, using languages such as the Semantic Web Rule Language (SWRL) [111].

Lastly, the *service grounding* maps an abstract service description – as provided by OWL-S – to a concrete service implementation. It addresses issues such as protocols and message formats, which are required for the actual invocation of a service. For that purpose, the OWL-S specification refers to WSDL and defines correspondences between OWL-S and WSDL constructs. These are depicted in Figure 2.8 [181].

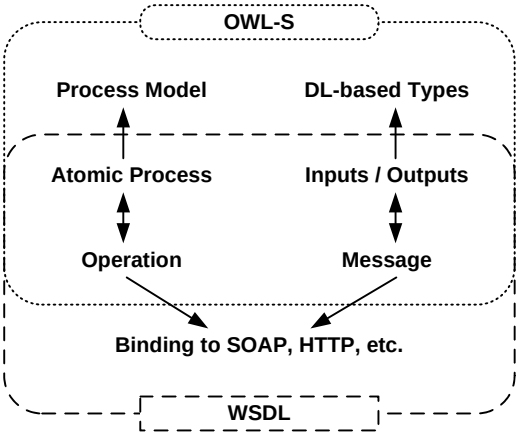


Figure 2.8: Mapping between OWL-S and WSDL (taken from [181])

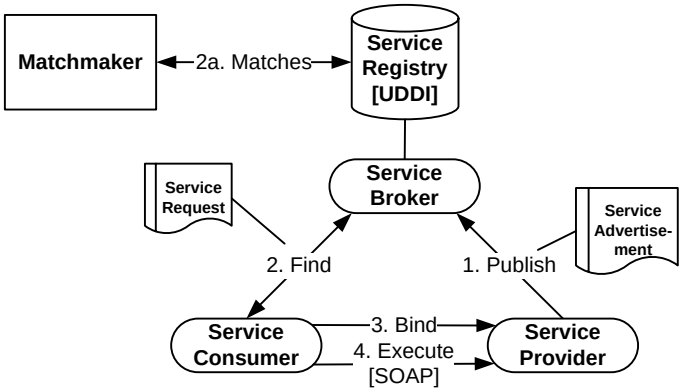


Figure 2.9: Service Discovery Reference Architecture, adopted from [69, 88, 247]

Service discovery is the process of finding appropriate services from a set of services, which could be provided in a central or distributed service repository [210]. After a set of appropriate services has been discovered, it is possible to *select* the most suitable service and bind/invoke it [90, 250].

Especially if regarding the automatic invocation and execution of Web services or even the (semi-) automatic composition of Web services to business processes and workflows, it is crucial to address the discovery of services. Only if appropriate services can be identified, it is possible to address further problems, such as QoS [25, 48, 74], and to select a service and include it into a workflow [47].

Based on the already known diagram from Figures 2.1 and 2.2, Figure 2.9 shows the main components of the service discovery process. Compared to Figure 2.2, three elements have been added respectively changed: First, the service consumer needs to formulate a *service request* in order to find appropriate services in the service registry. This service request can have different forms, e.g., be based on keywords (cp. Section 2.3.1). Second, the WSDL is substituted by the more generic term "service advertisement" related to the publication of service descriptions. Service advertisements can have different forms, amongst others the formerly used WSDL. Third, a *matching engine* has been added to the service registry. Matching engines take into account the service request and match it with the available service advertisements.

According to Tsetsos et al., the following factors affect the discovery process [247]:

- The ability of service providers to describe their services.
- The ability of service requesters to describe their requirements.
- The effectiveness of the service matchmaking algorithm.

The first two list items directly depend on the expressiveness of the used SWS formalism, i.e., to which degree the semantics of services can be formalized [154]. Analog to this listing, the following concepts are essential in service discovery:

Service Advertisements describe a service offer with respect to different aspects, i.e., functional capabilities, non-functional aspects etc. Depending on the type of the service registry adopted and the expressiveness of the Web service standard used, the service description may be more or less formal and detailed.

Service Requests declare the requirements of a service requester regarding functional, non-functional, and technical service capabilities. Service requests are formulated in concrete queries, which can have different forms (cp. Section 2.3.1).

Matching Engines (or *matchmakers*) match service advertisements and requests and calculate a DoM between them. The actual matching is a (pairwise) comparison of a service advertisement and a service request [90]. A matchmaker can be syntax- or semantic-based or be hybrid [3].

The DoM "can be formally defined as a value from an ordered set of values that express how similar two entities are with respect to some similarity metric(s)" [247]. This might be a numerical similarity value within a strictly defined interval or a value from a predefined set. In service matchmaking based on semantic information, logic-based *subsumption* relations between semantic concepts from an ontology are often employed as DoMs (cp. Chapter 3).

As it was mentioned in Section 2.2, a syntactic description of services is insufficient for service discovery as it could lead the matchmaking algorithm to wrongly assess a service offer to meet certain requirements [61]. This might be the case because services might have a syntactically similar, but semantically different description and vice versa [3, 146].

In this thesis, two of the abovementioned factors that affect the discovery process are addressed, namely the way service requesters describe their requirements (Chapter 4) and the conception of service matchmaking algorithms (Chapter 3), which are at the core of a matching engine. As SAWSDL has been chosen as primary SWS formalism, we concentrate on data and functional semantics (cp. Section 2.2.1), which is consistent with most of the related work (cp. Section 3.5). As a foundation, the next sections will give background information on query formulation for Web services (Section 2.3.1) and matchmaking (Section 2.3.2).

2.3.1 Query Formulation

Several approaches for service retrieval exist, which can be classified into different approaches regarding the kind of information given in a service request and, hence, available for matchmaking. In the following, these approaches are shortly introduced [118]:

Keyword-based retrieval is the most common query format on the Internet: The service requester specifies terms which are compared by a search engine with an item's description using string-matching techniques. In the work at hand, the terms would be compared to a service description. *Table*-based retrieval enhances keyword-based retrieval by allowing to define attribute-value pairs. For service discovery, it would be possible to define, e.g., a concept or keywords for certain service components like input and output messages [127].

Semantic-based retrieval allows the definition of queries based on semantic concepts instead of keywords. This way, it is possible to define a service request much more precisely and to derive subsumption or other relationships between semantic concepts defined in a service request and a service offer.

State-based retrieval enhances semantic-based retrieval by allowing to define pre- and postconditions for a service. Thus, it is possible to model the desired *behavior* of a service both in a service offer and in a service request. In contrast to semantic-based retrieval, which is mostly restricted to the *service signature*, state-based retrieval applies to the *service profile* (cp. Section 2.2.1).

In practical application, all mentioned approaches may be combined.

Concerning service registries like UDDI and ebXML, keyword- and table-based discovery approaches are commonly used. As it has been already discussed in Section 2.2, these syntax-based approaches are not sufficient to describe service capabilities and, hence, a service query. As another point, text-based descriptions rather include information on how to invoke a service than on its behavior. Thus, services may require and provide the same set of parameters while offering a different behavior. Although table-based approaches commonly perform better than keyword-based approaches, they do not necessarily lead to satisfying retrieval results [127]. Another drawback of these approaches are the missing relationships between the specified entities. For example, certain inputs and outputs may indeed be part of a single service described in WSDL, but can belong to different operations.

In contrast to keyword- and table-based approaches, semantic-based approaches consider the semantics of service elements, so that similarities can be detected even if there are syntactic differences. For this, such approaches make use of, e.g., subsumption relationships, which can be inferred by a reasoner from subsumption hierarchies of concepts defined in an underlying ontology. Typically, the concepts associated with inputs and outputs are analyzed or the classification of whole service documents is explored [118, 127]. In the subsumption-based approach by Paolucci et al., the authors consider a service advertisement as a match to a given service request, if all the inputs and outputs of the request are sufficiently

satisfied by the advertisement (cp. Chapter 3) [201]. The authors state that the simplest form of fulfilling this condition would be an exact match between the respective concepts. But since this condition is often too restrictive, they also allow for relaxed matching, i.e., the degree of similarity between an advertisement and a given service request is computed in form of a DoM (cp. Section 3.1.2).

Even though providing a richer query model than keyword-based retrieval, semantic-based matching is also not incorporating the behavior of services – two services, which have semantically identical input and output messages, may still lead to different results. This issue is addressed by state-based retrieval approaches, which include the specification of pre- and postconditions, which must hold before and after the execution of a service (cp. Section 2.2.1). In doing so, an abstract service comprises a set of state transformations [118]. A service advertisement matches a given service request, if it can be proven that the advertised service achieves the requested functionality. Since this implies a large complexity with respect to both, the modeling of the semantics and the computational effort of the proof process, the application of deductive approaches in practice faces serious difficulties [127]. For this reason, preconditions and effects have also been neglected while designing the successor of WSDL-S, namely SAWSDL (cp. Section 2.2.3). Nevertheless, they play a very important role in WSMO and WSMO-Lite [81, 251].

Apart from the information addressed in a retrieval approach, query formulation in terms of the formalism used to express requirements is also of interest. Different query formalisms for SWS are analyzed in Chapter 4.

2.3.2 Matchmaking

With regard to (semantic) Web services, matchmaking is the process of finding suitable service offers to fulfill a given service request [61]. Accordingly, a *matching engine* is a tool which implements such a process. Assume, for example, that a service consumer requires a service which provides financial information, more precisely, stock quotes. In a mature market of services, numerous service offers would be registered which supply stock quotes [203]. These offers differ in a number of dimensions which are related to the four different kinds of semantics defined by Sheth et al. (cp. Section 2.2.1) and also the number and type of message parameters. For instance, one service offer could require the stock's ticker symbol as input, while another requires its full name. Yet another service offer might allow specifying the name of the stock exchange from which the quote derives. Likewise, the number of output parameters can differ in the service offers. For instance, one service might return the number of traded stocks or the date and time along with the latest quote. Another service might return the ticker symbol to allow for verification.

An example is depicted in Figure 2.10. The task is to find an optimal service that provides stock quotes. The service request is depicted on top. In this example, the request is described using syntactic names and descriptions and semantic concepts describing the inputs and outputs of the desired service. The service offers are illustrated at the bottom.

First of all, each service has inputs and outputs syntactically specified by their names. The corresponding semantic annotations are given in parentheses, with multiple annotations separated by commas. Second, service offers might differ in a number of non-functional dimensions, such as cost of invocation or QoS (not shown in the figure). One service offer, for instance, might only charge 1 cent per invocation, but does not make any QoS assurances. Another service offer might charge 5 cent per invocation, but assure 99.999% availability, along with a maximum 500 ms response time. Third, service descriptions might contain information about preconditions and effects, i.e., the *service specification* (not shown in the figure). The service specification contains information about the initial state – i.e., the state of "the world" before a service is invoked – and a goal state which describes the state of "the world" after the service has been invoked [61].

In the context of this thesis, the focus will be on the first dimension, which is functional (and semantic) matchmaking between services, but does not regard the service specification; i.e., we apply semantic-based, stateless retrieval as presented above. The service specification has to be omitted as there is still no agreement how it could be included in SAWSDL. This implies that there is no commonly accepted formalism usable to describe preconditions and effects.

We assume that a number of service offers have been registered at a central registry. Given a service request, matching services are to be identified. The offers are to be ranked with respect to their relative similarity to the service request. In this context, matchmaking can be divided into three distinct steps.

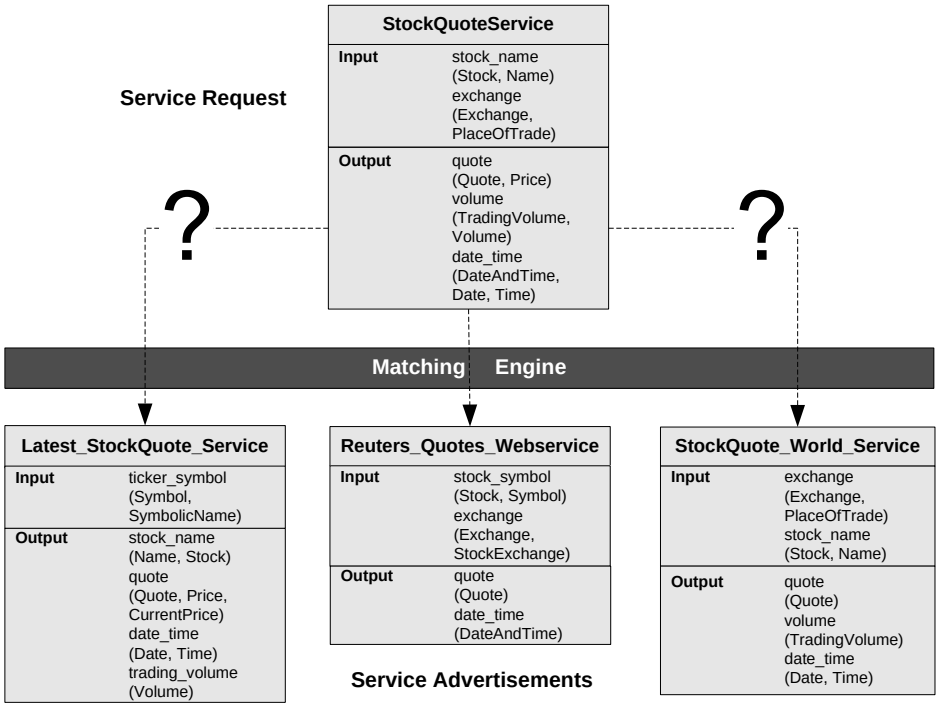


Figure 2.10: Web Service Matchmaking (Example)

Identify the data items to be matched: In a first step, it is necessary to define which components and information of a Web service description will be matched. As it will be shown in the next chapter, the variety of information that could be incorporated in the matchmaking process ranges from referenced semantic concepts to structural and syntactic information. Of course, the decision which data elements will be exploited depends on the Web service standard regarded.

Measure similarities: After the data objects to be matched have been identified, it is necessary to measure their similarities. In the case of SAWSDL-based services, this primarily concerns the similarity between interfaces, operations, or message parameters. Again, the choice of similarity measures is broad and ranges from subsumption relations to syntactic name similarity and hybrid approaches.

Matching: Assigning similarities to pairs of components is one step. Another one is to actually find the best matching components, i.e., to associate each component in the service request with the best matching component in the service offer. This is commonly addressed to be an *assignment problem* [41].

The analysis how these three steps should be addressed is the content of Chapter 3. As it will be further explained in Section 3.5, there are several researchers experimenting with service discovery based on semantics. In anticipation of the presentation of the different matchmaking approaches in Chapter 3, semantic matchmaking approaches can be classified based on the following aspects [130]:

Logic- vs. non-logic matching: On the one hand, matchmaking can address semantics and derive logical inferences or assess other relationships between them. On the other hand, techniques from the field of IR could be used in order to make use of syntactic service descriptions. Furthermore, these approaches can be combined, leading to hybrid matchmaking.

Service standards: Most matchmakers address exactly one particular Web service standard, mostly OWL-S, SAWSDL, or WSML. Other matchmakers apply to a formulation of service capabilities in another format, i.e., DL.

Service elements: Service descriptions are made up from different elements; matchmakers can be distinguished by the elements they exploit for detecting the similarity of service requests and offers. Usually, this is the service signature or service specification.

Of course, these three aspects can be combined in various ways – a set of service elements usable for matchmaking OWL-S-based services might not be suitable for SAWSDL etc.

Based on the theoretical and technical background provided in this chapter, Chapters 3 and 4 present the contributions of this thesis in terms of matchmaking approaches respectively query formalisms for SWS.

Part II

Adaptive Matchmaking and Query Formulation

3 Self-Adaptive Matchmaking for Semantic Web Services

As it was explained in Section 2.3.2, matchmaking is one of the core components in the service discovery process and might rely on semantic or syntactic data. In the work at hand, two matchmakers for SAWSDL are designed, implemented, and evaluated. To some extent, these matchmakers share the same techniques, but especially the measurement of similarities is based on different ideas and assumptions. The first matchmaker, Logic-based Matchmaking Approach for Semantic Web Services (LOG4SWS.KOM), is based on "classic" subsumption matching and introduces an intuitive way of weighting and combining different subsumption matching degrees. Furthermore, a linguistic-based fallback strategy is proposed in order to meet the fact that a semantic service description is not necessarily complete. The second matchmaker, Coverage-based Matchmaking Approach for Semantic Web Services (COV4SWS.KOM), is based on metrics from the field of semantic relatedness in ontologies. This way, COV4SWS.KOM incorporates more fine-grained semantic-based relationships than conventional subsumption matching-based approaches. Additionally, COV4SWS.KOM introduces the adaptation to varying quality of syntactic descriptions and semantic annotations at different levels of service abstraction.

The following hypotheses are applied in order to determine the outcome of the work conducted in this chapter. These hypotheses will be discussed in Section 3.6.

H1: The automatic derivation of numerical equivalents of subsumption DoMs by statistical means allows to customize a matchmaker regarding different basic assumptions. This especially concerns different presumptions what a semantic annotation on a distinct service abstraction level actually means as well as how the subsumption relationships between semantic concepts are actually defined.

H2: In service discovery, information from all service abstraction levels should be regarded. The weighting of these levels should be based on the impact of the syntactic and semantic description on service discovery results, i.e., their quality and usefulness.

H3: Apart from subsumption-based similarity determination, metrics from the field of relatedness measurement of semantic concepts in ontologies might be helpful to compute the similarity between a service request and a service offer.

This chapter is structured as follows: As an introduction, we will identify and discuss shortcomings of current approaches to SWS matchmaking. Furthermore, requirements for service matchmakers, which could be feasible to overcome the discussed issues are proposed. Based on these findings, Sections 3.2 and 3.3 present the concepts, specifications, and implementations of LOG4SWS.KOM and COV4SWS.KOM. Both matchmakers are extensively evaluated in Section 3.4. The evaluations contain a section on the performance in terms of IR metrics and runtime performance, as well as a discussion of the benefits and drawbacks that arise from the application of the proposed matchmakers. This chapter closes with an overview of the related work (Section 3.5), and conclusions (Section 3.6).

3.1 Problem Statement and Specification of Requirements

Especially if regarding the vision of automated service invocation [183], it is necessary to detect services which provide the right capabilities in a very precise way, i.e., only those services should be found that provide the requested capabilities. As it has already been described in Chapter 2, syntax-based service standards like WSDL do not provide the means to describe service capabilities in enough detail [247]. This does not only apply to the functional and non-functional description of a service, but also to data types [42]. Shortcomings of syntax-based service description languages regarding service discovery is one of the prime reasons for the integration of semantic information into Web service descriptions and has been regarded from the very beginning of SWS research, e.g., [28, 164, 201, 241].

Of course, the mere existence of semantic descriptions does not improve service discovery – the semantic information need to be incorporated during the matchmaking process. In order to serve this purpose, researchers have proposed a multitude of matchmaking approaches (cp. Section 3.5) [130]. As is evident

from the results of the Annual International Contest S3 on Semantic Service Selection – Retrieval Performance Evaluation of Matchmakers for Semantic Web Services (S3 Contest) – which serves as an annual contest to compare and discuss matchmakers for different service formalisms – the best matchmakers (in terms of IR metrics like precision and recall) combine logic- and non-logic approaches into *hybrid* matching [142]. Purely logic-based approaches result mostly in mediocre matchmaking results and cannot keep up with their hybrid counterparts. On the one hand, especially the incorporation of syntax-based similarity values involves often some degree of uncertainty, as these methods are, e.g., unaware of different meanings a term might have [160, 176]. On the other hand, logic-based matching (and semantic-based matching in general) defines clear rules which makes it easy to follow and recapitulate matchmaking results. In the work at hand, the usually applied subsumption matching approaches are modified or replaced by other semantic-based similarity metrics in order to facilitate competitive matchmaking results without applying non-semantic-based similarity measures. However, syntax-based similarity measures are used as a fallback strategy.

In the following, we will analyze shortcomings and potential improvements of current service matchmaking approaches. This analysis is based on the three matchmaking steps presented in Section 2.3.2, namely the identification of data items that will be incorporated in matching (Section 3.1.1), the actual measurement of similarities between different elements (Section 3.1.2), and the determination which elements from a service offer best matches an appropriate element from a service request (Section 3.1.3).

3.1.1 Identification of Data Items to be Matched

As presented in Section 2.1.1, WSDL possesses an *abstract* part which defines what a service does. This abstract definition is made up from *interfaces* and corresponding *operations* and input and output *messages*, which are usually defined using XSD types. A couple of different approaches exist on how to integrate and combine similarities on these different levels into an overall similarity value for the whole service. In quite a number of approaches, only certain elements (in the majority of cases the service signature, i.e., inputs and outputs) are used. The restriction to certain service abstraction levels might be suitable if regarding that the evaluation of service matchmakers is today based on service descriptions, where all possible service elements should be defined (cp. Sections 2.3.1 and 3.5). However, if service queries in terms of, e.g., an SQL statement, are defined, a matchmaker needs to be able to incorporate all information defined in a particular query. As it is most likely that a human user would search for a certain service functionality or service domain, which is usually defined on interface and operation level, a restriction to, e.g., the service signature is not sufficient.

As a result of the restriction of many matchmakers to the service signature, the aggregation of similarity values from different service abstraction levels is often not explicitly regarded. In fact, in WSDL, the desired functionalities a service requester is looking for, are provided by operations. Ideally, for each requested operation, a matching operation in a service offer should be identified. In line, the overall similarity of two services should relate to the degree to which their respective operations match, i.e., to which the offer provides the requested functionality.

Another question is how to combine the different similarity measures for every service abstraction level. This depends on how useful the element description on every level is: If a semantic or syntactic description is completely missing on interface and operation level, it seems to be likely that the overall similarity value for the service should heavily depend on the message level. However, it might be the case that the interface and operation *names* are sufficient enough for matching. It seems to be unlikely to derive an universally valid way to combine the different service abstraction levels.

As it was mentioned before (cp. Section 2.2), SAWSDL does not specify any semantics by itself; which parts of a WSDL needs to be semantically annotated could however be defined by a SAWSDL framework or a service registry. Two antipodes regarding the richness and quality of semantic annotations for Web services are thinkable: In a company-internal service registry with a manageable number of services, every service abstraction level might be perfectly described by semantic concepts – hence, a purely semantic-based matchmaker might be adequate. In other cases, especially public service registries on the Internet, it is quite likely that the better part of all service descriptions is not semantically annotated. This assumption is backed by current observations: At the time of writing, *seekda!*[1], as one of the

[1] www.seekda.com

world's largest Web services search engines, lists about 28,000 different Web services with the number quite stagnating in the last 18 months. The number of semantic Web services is about 410 without the existing scientific test collections (cp. Appendix B.2.3) and about 3500 including these collections [142]. Of course, seekda! only lists all publicly available services, i.e., Web services that are provided using an access-restricted service repository will not be listed. It is quite likely that the majority of SWS will not be publicly available, but advertised in company-internal repositories [135].

It seems to be reasonable to make the combination of different service abstraction levels dependent on the service set that comes into consideration. Therefore, it is necessary to provide weights for the different service abstraction levels in the first place. As an enhancement, this weighting could be conducted automatically based on the impact a particular level will have on the results of the matchmaking process.

Constraining the matchmaking to semantic concepts seems to be impractical. In an ideal world, all service elements would be completely described using semantic concepts [160]. However, it is quite likely that there is, e.g., no ontology that could be used in order to annotate all elements of the service description or the service provider is just unaware of it [77, 103, 244]. Thus, there should be some kind of fallback solution that could be applied if a semantic-based description is missing or cannot be processed.

Summarized, this leads to the following requirements a matchmaker should fulfill regarding the identification, usage, and combination of data items a service might offer:

- As different service abstraction levels provide different kinds of more or less useful information about a Web service, it should be possible to include all levels in the matchmaking process or exclude single levels.
- As operations encapsulate desired functionalities, matchmaking should focus on operations instead of service interfaces.
- A matchmaker should be able to adapt itself to different degrees of service description usefulness on different service abstraction levels.
- A matchmaker should be able to make use of syntactic information if there is no usable semantic description of distinct service components.

3.1.2 Measurement of Similarities between Objects

Similarity measures are at the core of every matchmaker implementation. They assess the relative similarity between two components, i.e., two elements of a Web service description. In the case of semantic Web services, this commonly concerns the similarity between a pair of interfaces, operations, or message parameters. The actual measurement of similarity values is usually based on semantic annotations or syntactic information. Frequently, different types of measures are also combined into a unified measure, or linguistic measures are used as a fallback strategy once semantic information is not available (e.g., [212, 244]). Furthermore, it is possible to make use of syntax-based retrieval methods in order to exploit the *implicit* semantic description, e.g., by applying syntax-based algorithms to the names of unfolded semantic concepts (cp. Section 3.5). As a third option, structural similarity can also be derived [140, 212].

While the incorporation of syntax-based similarity values often involves some degree of uncertainty, as these methods are, e.g., unaware of different meanings a term might have [160, 176], purely logic-based approaches often result in mediocre matchmaking results. This might be traced back to the fact that many ontologies are rather taxonomies, which do not provide the rich semantic concepts needed for sophisticated logic-based matching [132, 255]. Hence, the first aspect worth observing regarding the measurement of similarities should examine alternative similarity measures for SWS matchmaking.

As a second major concern, the representation of similarity measures should also be regarded: Similarity measures either produce result on a discrete or continuous scale. Discrete means that the measure's output comes from a finite set of possible outcomes – here, one example are the DoMs defined by Paolucci et al. (see below). In contrast, continuous means that the similarity value can take an arbitrary number from a limited – or possibly unlimited – range of values, e.g., [0..1] or [0..∞[.

As a major disadvantage, a discrete scale does not allow any further distinction between matches that fall into the same class. Thus, it only permits an insufficient ranking of services, even though more

detailed matching information might be available [82]. In order to tackle this issue, many matchmaking approaches first use a discrete scale of DoMs and afterwards use a second similarity measure to rank the services in the particular DoM category (e.g., [140]). Also, a discrete scale does not allow the easy combination of different measures into a single result. However, the latter is preferable again when it comes to ranking of services. Furthermore, many algorithms that are suitable to find the best matching components (see next section) rely on the numerical definition of "costs", which makes a continuous scale essential.

While a number of similarity measures have been introduced, a dominant fraction of matchmaking approaches still relies on a discrete scale to represent the DoM between a pair of services. Commonly, this is a four-level scale as originally proposed by Paolucci et al. [201], usually applied with some extensions (e.g., [119, 164, 245]).

On the other side, a problem with a continuous scale is that it does not permit any statement about the underlying semantic relation between services, i.e., if a service offer is more generic or more specific than the other. The scale solely states their overall relative equivalence or similarity. Another drawback concerns the fact that a continuous scale must be derived from a discrete scale if regarding, e.g., subsumption-based similarity "classes". Ambiguity is necessarily contained in the mapping between discrete and continuous DoMs, because a certain logic subsumption relation does not naturally relate to a given relative equivalence. For example, Liu et al. assign a similarity of $\frac{\alpha}{1+\alpha}$ to a subsume relationship for inputs and $\frac{\alpha}{2+\alpha}$ to a plugin relationship for inputs without further explaining why these values have been chosen [169]. Fernández et al. and Guo et al. arrange for the usage of numerical values but do not specify how these values should be calibrated [82, 97]. Ideally, there should be a justified relationship between numerical similarity values and discrete DoMs, depending on the Web services regarded.

Usually, there is a predetermined order of DoMs – e.g., Paolucci et al. define this as *exact* > *plugin* > *subsumes* > *fail* (cp. Section 3.5) [201]. Paolucci et al. reverse the meaning of plugin and subsumes for outputs and inputs – for former, a plugin match specifies that the class used in a service offer is more *generic* than this used in a service request. For inputs, a plugin match indicates a more *specific* class used in a service offer than in the service request. As a result, the ranking is practically reversed for the rating of more specific and generic DoMs for inputs and outputs [201]. While the grading of fail and exact as the best and worst DoM is unambiguous, there are different proposals on how to rate the DoMs plugin and subsume: According to Paolucci et al., service offers that deliver a more generic output than requested should be ranked higher than those service offers that deliver a more specific output than requested [201]. The underlying idea is that the first service's output includes *all* desired items, while the second service's output only contains a *part* of them. In case of inputs, Paolucci et al. use the opposite assumption – service offers that accept a more specific input, as compared to the service request, are favored over server offers that expect a more generic input.

Bellur et al. and Cardoso reverse this ranking regarding *subsumes* and *plugin* respectively invert the meaning of these DoMs, which leads to an inversed ranking [17, 45]. Cardoso argues that a more generic *input* (i.e., a subsumes relationship as defined by Paolucci et al.) is favorable, because it will certainly accept the input which has been specified in the service request. Again, in case of *outputs*, the ranking of DoMs regarding more specific respectively more generic classes is inverted: Service offers with a more specific output are favored over those with a more generic output. Here, Bellur et al. argue that Paolucci's assumption that a service provider who advertises a certain output (i.e., a semantic concept describing the output) will deliver *every* subclass of this output, is not necessarily the cased and it is more realistic that only *some* of these subclasses will be supported.

As Bellur et al. show, the scheme by Paolucci et al. encourages an imprecise – namely, overly generic – annotation of outputs [17]. In detail, service providers would be given incentive to annotate their services' outputs with the *owl:Thing* concept, because it constitutes the most generic concept in an OWL ontology. In line, Cardoso's scheme favors the most generic annotation of input parameters.

An example is depicted in Figure 3.2; it relates to the sample ontology from Figure 3.1, which depicts an excerpt from the *books* ontology in SAWSDL-TC (cp. Section B.2.2). Assume that a service consumer requests a service that delivers *Monograph* as output. However, such a service with exact this output is not available but there are two service offers with related output: The first service offer provides a more *generic* output, namely *PrintedMaterial*. According to Paolucci et al., the service provides all *Monographs* as output, which is desired. However, its output also contains undesired concepts, e.g., *Serial-Publications*. In contrast, the second service offer provides a more specific output, namely *Novel*, which is a subconcept

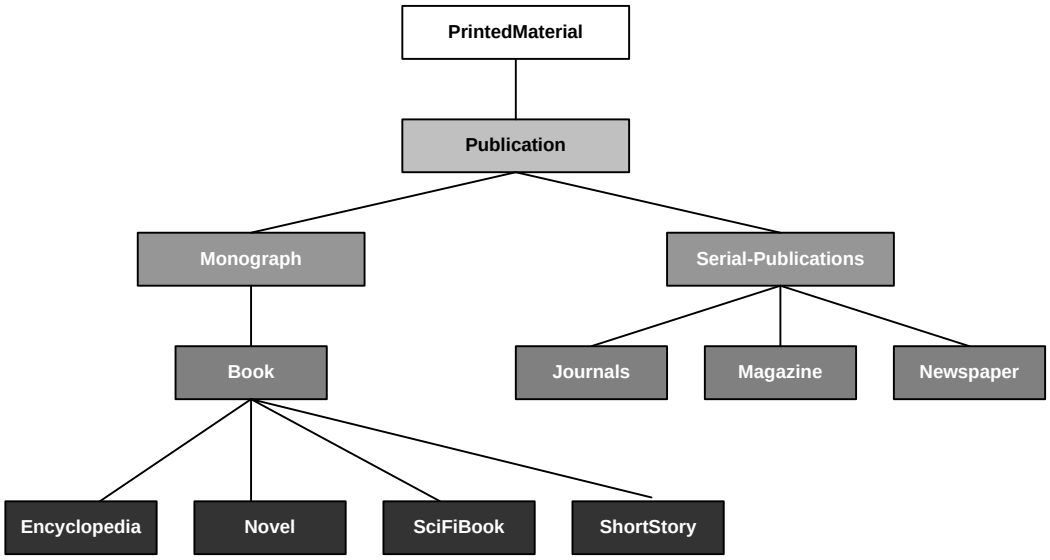


Figure 3.1: Example Ontology

of the desired *Monograph*. However, it does not contain any of the *Encyclopedias* the service consumer would like to receive as output, too. Alternatively stated, the first service achieves maximum recall at the cost of lower precision, while the second service achieves maximum precision at lower recall. To deepen the understanding of aforementioned schemes, let us consider two exemplary service requests (A, B) and four corresponding service offers (A1, A2, B1, B2). Their input and output parameters, along with a short description, are provided in Table 3.1. The resulting ranking, according to the previously described schemes, is provided in Table 3.2.

Both ranking scheme approaches have their justification, depending upon which assumption regarding the generalization and specialization of semantic concepts in an ontology holds true. Hence, it would be useful to derive the ranking of these DoMs automatically. This applies to SAWSDL in particular, as it does not define the "meaning" of semantic annotations on different service abstraction levels.

All things considered, the following requirements regarding the deduction of similarity values and their representation have been identified:

- A matchmaker should be able to represent DoMs in a way that allows the easy combination with other similarity values. In most cases this means that similarity values should be arranged on a continuous scale.
- A mapping from a discrete to a continuous matching scale should be justified and depend on the set of services (and hence the basic assumptions regarding the meaning and relationships between semantic concepts) a matchmaker is applied to.
- If applying subsumption matching, a matchmaker should be able to rank DoMs based on the assumptions modeled in an ontology.

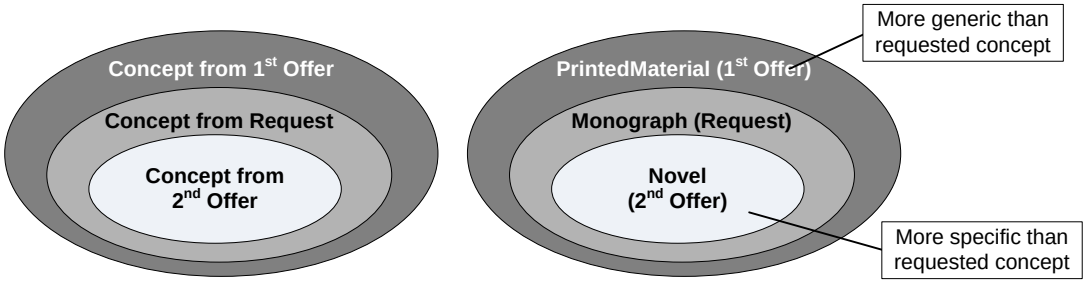


Figure 3.2: More Generic and More Specific Parameters in Comparison to Requested Parameters

Table 3.1: Exemplary Service Offers and Requests

Service	Input	Output	Description / Comment
Req. A	Author	**Monograph**	Provides monographs written by a given author.
Off. A1	Author	**PrintedMaterial**	Provides a more generic output than specified in A.
Off. A2	Author	**Novel**	Provides a more specific output than specified in A.
Req. B	**Monograph**	Author	Provides the author for a given monograph.
Req. B1	**PrintedMaterial**	Author	Expects a more generic input than specified in B.
Req. B2	**Novel**	Author	Expects a more specific input than specified in B.

Table 3.2: Ranking of Service Offers According to Different Subsumption Level Schemes

Request	Scheme	Ranked Higher (DoM)	Ranked Lower (DoM)	Result
Req. A	Paolucci et al. [201]	Off. A1 (more generic output; *plugin*)	Off. A2 (more specific output; *subsumes*)	A1 > A2
Req. A	Cardoso [45]	Off. A2 (more specific output; S_i case 3, =1)	Off. A1 (more generic output; S_i case 2, <1)	A2 > A1
Req. B	Paolucci et al. [201]	Off. B2 (more specific input; *plugin*)	Off. B1 (more generic input; *subsumes*)	B2 > B1
Req. B	Cardoso [45]	Off. B1 (more generic input; S_o case 2, =1)	Off. B2 (more specific input; S_o case 3, <1)	B1 > B2

3.1.3 Matching Service Components

The matching of components is often based on the *greedy approach* presented by Paolucci et al. [17, 201]. In general, greedy algorithms are simple approaches to solve optimization problems and rely on the assumption that a global optimum can be found by making the locally optimal choice for each step of the optimization approach. Of course, this is not necessarily the case, hence the greedy approach may lead to suboptimal results [55]. Regarding service matchmaking, the greedy approach determines for each concept from a set of elements (e.g., all input parameters from a service request) a corresponding concept in a second set (e.g., all input parameters from a service offer) to which it has a *maximum* DoM. As Bellur et al. show, the greedy algorithm suffers from a number of drawbacks in practical application [17]. For instance, the authors demonstrate that the algorithm's results can depend on the order of the components in the first set.

Given the presented drawbacks of the greedy algorithm, Guo et al. [97] were the first to publish a matchmaking algorithm based on *bipartite graphs*, so that matching can be traced back to general *assignment problems*. In this approach, the two sets of components to match each constitute one partition of nodes [41]. The edge weights then correspond to the similarity between the two nodes (i.e., components) that they are connecting. This way, *maximum weight bipartite graph matching* can be applied, which allows to make use of algorithms which solve the *linear sum assignment problem*. For this problem, a number of approaches have been proposed which aim at *minimizing* the edge weights while in the scenario at hand, it is necessary to *maximize* the average edge weight. However, this is not a real issue as

it is possible to transform a maximization to a minimization problem. Hence, the well-known *Hungarian* algorithm (or *Kuhn-Munkres* algorithm) can be applied to calculate an optimal, pairwise assignment between components [150, 151, 189]. Lawler has presented an implementation of the Hungarian algorithm which solves the bipartite graph assignment problem for a complete graph with two node sets of equal cardinality n within the complexity bound $O(n^3)$ [41]. An enhancement of the Hungarian algorithm for graphs with differing cardinalities has been presented by Bourgeois and Lassalle [35]. In the work at hand, we will make use of these enhancements of the original algorithm.

Regarding the approach presented by Guo et al., the matching is not based on the notion of a *global DoM* [97]. In line with Klusch et al. [140] the global DoM is perceived as a guaranteed fixed lower bound for the overall service compatibility; hence, the minimum DoM should be maximized. Instead, in the approach by Guo et al., the sum of edge weights (i.e., of similarity) is maximized. In some scenarios, this may lead to a suboptimal assignment with respect to the concept of a global DoM. To circumvent this problem, Bellur et al. present a modified version of the bipartite graph matching approach [17, 20]. In detail, the edge weights are modified in such manner that the matching standard algorithm maximizes the minimal DoM. This objective exactly corresponds to the concept of global DoM. Unfortunately, their scheme is not applicable in conjunction with continuous, but only discrete DoM scales.

As an alternative, Plebani and Pernici, for instance, simply utilize continuous DoMs in conjunction with bipartite graph matching [212]. Accordingly, they assume that the global DoM is yielded by the average of edge weights, not by the worst individual match. Again, this does not guarantee a global DoM. However, this is more a problem of the basic assumptions regarding service matchmaking. Despite the allure of large service marketplaces where numerous similar services are advertised by different vendors and almost every possible service is provided, such a market is far away from being realized. As long as the number of services is not radically increased, a state where it will be possible to automatically discover and invoke services is not realistic. In many application areas, it might never be the case that services exist which can suit most of the service requesters' needs. As a result, service requesters will have to adapt their own applications in order to make use of services. Considering this, it makes sense to identify services, which are similar to a service request, without guaranteeing a fixed lower bound for the overall service compatibility. Notably, the matchmaking approach by Plebani and Pernici, which is currently providing the best SAWSDL matchmaking results in terms of precision, does not guarantee a certain DoM [212]. Analog to the approach used by these researchers, the matchmakers presented in this thesis also evaluate the *similarity* between Web services – a lower bound of service compatibility is not given. Hence, we will be able to make use of the abovementioned Hungarian algorithm for matching.

While the application of bipartite graph matching has proven its usability, there remains one last issue: Usually, for every element in a service offer, a corresponding element in a service request should be available. Some researchers argue that a mismatch in parameter cardinality between service request and offer might imply that the offer is unsuitable for the specified purpose. Paolucci et al., for example, state that every output parameter in the service request must be satisfied by an output in the service offer and every input parameter in the service offer must be satisfied by an input in the service request; else, the offer is assumed to fail for the given request [201]. The same requirement is stated in, e.g., [17, 115, 140]. Such a behavior makes sense if following an approach that defines the minimum DoM as the global DoM and thus guaranteeing a certain DoM. However, if relaxing the demand of the minimum DoM as a lower bound, service offers which do not meet the cardinalities of the service requests should also be regarded.

By an example, we will demonstrate that the same type of mismatch in cardinality does not necessarily allow a general statement regarding a service offer's suitability to fulfill a request. For that purpose, we define two simple example services:

- *HotelService*, which finds one or more hotels based on specified criteria.
- *SSNService*, which finds a person's social security data based on specified personal data.

In the following, let In_r and In_o denote the set of inputs for the requested and offered service respectively; accordingly, let Out_r and Out_o denote the set of outputs. If the sets are equivalent for service request and offer, the generic expressions *In* and *Out* will be used for inputs and outputs respectively. $|S|$ denotes the cardinality, i.e., number of elements, in a set S.

Table 3.3 summarizes a number of scenarios where input or output parameter cardinality differs. The second column states the underlying service and general type of cardinality mismatch. The third column

Table 3.3: Example Services with Differing Parameter Cardinalities

Example No.	Service Name and Type of Cardinality Mismatch	Parameter Sets	Comment
1	HotelService $\|In_r\| > \|In_o\|$	In_r = {MaxPrice, ZIPCode, MinRating} In_o = {MaxPrice, ZIPCode} Out = {Hotel}	Problematic, because the offered service does not allow restrictions regarding the hotel rating.
2	SSNService $\|In_r\| > \|In_o\|$	In_r = {FirstName, LastName, DateOfBirth} In_o = {Name, DateOfBirth} Out = {SocialSecurityNumber}	Not problematic, because the first and last name can simply be concatenated to generate the full name.
3	HotelService $\|In_r\| < \|In_o\|$	In_r = {MaxPrice, ZIPCode} In_o = {MaxPrice, ZIPCode, MinRating} Out = {Hotel}	Not problematic, if the minimum rating has a default value.
4	SSNService $\|In_r\| < \|In_o\|$	In_r = {Name, DateOfBirth} In_o = {FirstName, LastName, DateOfBirth} Out = {SocialSecurityNumber}	Potentially problematic, because the name has to be split into the first and last name.
5	SSNService $\|Out_r\| > \|Out_o\|$	In = {SocialSecurityNumber} Out_r = {FirstName, LastName} Out_o = {Name}	Potentially problematic, because the first and last name have to be parsed from the name.
6	SSNService $\|Out_r\| < \|Out_o\|$	In = {SocialSecurityNumber} Out_r = {Name} Out_o = {FirstName, LastName}	Not problematic, because the first and last name can simply be concatenated to the name.

contains the actual parameter sets; a comment is included in the fourth column. As Table 3.3 demonstrates, a given type of mismatch in parameter cardinality does not allow a general statement about a service offer's suitability to fulfill a given request. For instance, the service offer in the third example could well be suitable to serve the requested purpose if a default value for the *MinimumRating* input exists, such that it does not constrain the service output. Likewise, the service offer in the fourth example could easily be used if it was possible to parse the *Name* input parameter into *FirstName* and *LastName*. In both scenarios, however, a more sophisticated approach beyond subsumption reasoning is required. For instance, in conjunction with the third example, an additional evaluation of the syntactic Web service description, namely the associated messaging schema, would be required.

While this example addresses parameters, similar considerations can be made for operations and interfaces. E.g., if the number of operations demanded in a service request exceeds the number of operations in a service offer, this offer might still be helpful to some degree.

All things considered, we have identified two issues with current matchmaking approaches respectively requirements the actual matching should fulfill:

- Matching algorithms should avoid the shortcomings of the greedy approach. As this has already been covered in the related work, we will make use of the established Hungarian algorithm.
- Different cardinalities of service components should be addressed.

3.1.4 Recapitulation

Summarized, open issues in service matchmaking arise especially from the identification and combination of service elements to be matched and the derivation of similarity values. As this leaves a large room for improvements, these two aspects will be in the focus of the matchmakers presented in the next sections. In contrast, regarding the third aspect (matching of components), an already existing approach will be employed, namely bipartite graph matching using the Hungarian algorithm as enhanced by Lawler respectively Bourgeois and Lassalle, and implemented by Nedas [35, 41, 194].

LOG4SWS.KOM primarily aims at new ways to combine, rank, and weight similarities based on the classic subsumption DoMs by Paolucci et al. [201], while COV4SWS.KOM deals with an alternative way to compute similarities for service elements and adapt to the overall service similarity, based on the usefulness of information given on every service abstraction level.

3.2 LOG4SWS.KOM: Self-Adapting Semantic Web Service Discovery for SAWSDL

This section is structured as follows: In Section 3.2.1, requirements towards LOG4SWS.KOM arising from the analysis in Section 3.1 will be recapitulated and further specified. Afterwards, the fallback strategy, which is applied if semantic annotations are not available or cannot be processed, and a caching mechanism used to improve the matchmaking runtime are presented. In Section 3.2.3, the derivation of a numerical representation of discrete DoMs based on Ordinary Least Squares (OLS) is presented. This section ends with an overview of the implementation. The evaluation is described in Section 3.4.

3.2.1 Specification

The basic idea of the matching approach used in LOG4SWS.KOM is the derivation of logical and ontological relationships between elements specified in a service request and available service offers. Possible DoMs are, e.g., *exact*, *plugin*, *subsumes*, and *fail* (cp. Sections 3.1 and 3.5) with further DoMs proposed, e.g., by Klusch et al. or Tran et al. [136, 245]. LOG4SWS.KOM enhances this approach by applying matching to different service abstraction levels, integrating a fallback strategy, and deriving the overall similarity not by a lower bound, but as an average number. The overall similarity value is based on a continuous similarity measure that allows for flexible combination with other similarity values.

Selection of Matching Levels

As discussed in Section 2.2.3, different components of the *abstract* part of a WSDL document can be annotated using SAWSDL. For WSDL 2.0, these service components are interfaces, operations, and message parameters defined using XSD (cp. Section 2.2.3). For an overview, please refer to Figure 2.6, which shows the different levels of abstraction in SAWSDL.

Notably, inputs and outputs do not carry any semantic annotations according to the SAWSDL specification. Therefore, we do not consider them as separate abstraction levels. Instead, all parameters as defined by the XSD part of the WSDL document are mapped into a flat hierarchy and directly associated with the corresponding operation component. In detail, we recursively resolve the data structure of the XSD elements which are referenced by an operation's input and output component. In the course of this process, solely *simpleType* and *complexType* nodes are regarded as parameters. The approach is extensively described in the context of the ATSM in Appendix A.1.3.

Generally, a matchmaking process which includes all abstraction levels may either be conducted in a top-down or a bottom-up manner. Bottom-up means that the matchmaking is initially conducted on the lowest level, i.e., message parameters. The results are propagated to the level of operations. Consecutively, operations are matched, and the result is again propagated to the level of interfaces, where the final matchmaking is conducted. Depending on the concrete matchmaking algorithm, annotations on higher levels may be ignored. Alternatively, they may be used for additional matching and merged with results from lower levels. In a top-down process, matchmaking starts at the highest level, i.e., at interface level. It is possible that such processes terminate if a service request and service offer are equivalent at a

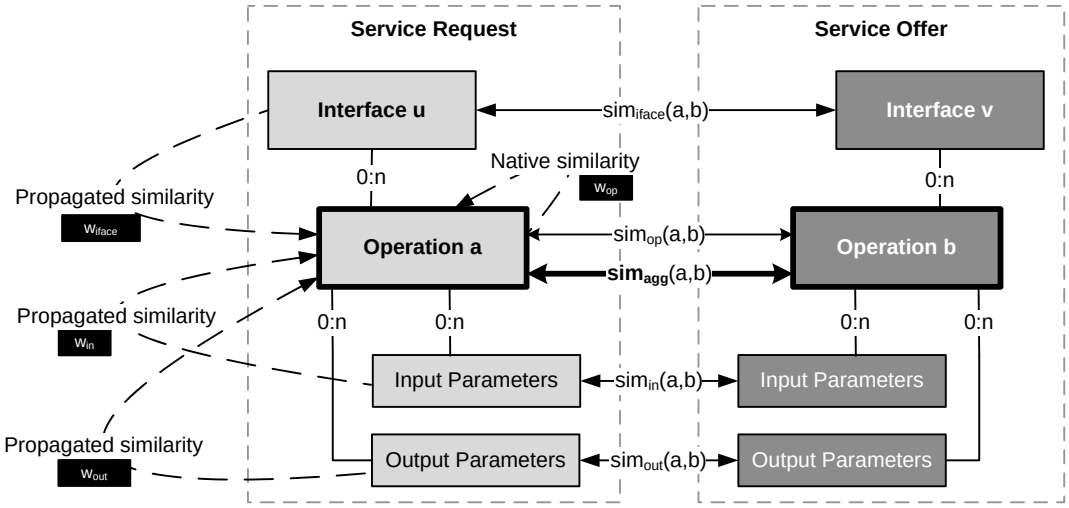


Figure 3.3: Matchmaking Process in LOG4SWS.KOM

certain level of abstraction. I.e., matchmaking on lower levels would be conducted only if services differ on higher abstraction levels.

Both, top-down and bottom-up matchmaking, have their advantages and drawbacks. Generally, top-down matchmaking can be very efficient, because the number of matching steps might be comparatively small. Yet, this requires that services are precisely annotated at the interface and/or operation level. Also, if the matchmaking process terminates on a higher abstraction level, there is no assurance that service request and offer actually match on the level of parameters. According to the SAWSDL specification [79], semantic annotations on interface level might provide a categorization which could be, e.g., a business area which is a quite broad characterization. Hence, restricting the matchmaking process to the interface level does not seem to be sufficient. For operations, the SAWSDL specification states that model references could address a high level description of the operation [79]. It can be seen that the decision if matching on interface and operation level is sufficient arises from the considered domain. If regarding a clearly arranged, relatively small domain, e.g., a particular industry with a commonly accepted domain model, matching on operation level might be sufficient, in other circumstances, it seems that the signature should also be regarded.

In the case of bottom-up matching, in contrast, this issue does not arise. However, matching on the level of message parameters can potentially be very expensive in terms of runtime performance, because the number of comparisons between service request and offer is significantly higher, depending on the respective parameter count. Also, it is unclear how matching results from different levels of abstraction should be consolidated. For instance, consider a pair of services that perfectly matches on the signature level. However, they significantly differ on the level of operations. The decision whether the service offer is suitable to satisfy the request is then ambiguous.

LOG4SWS.KOM attempts to unite the best of "both worlds" (i.e., top-down and bottom-up matching) by focusing on the *center* level of abstraction in a service, namely on operations. The overall similarity of two services relates to the degree to which their respective operations match, i.e., to which the offer provides the requested functionality. A similar approach is, for instance, applied in the Web service search engine *Woogle* (cp. Section 3.5).

A central challenge lies in efficiently measuring the similarity between pairs of operations. For that matter, we have devised an approach that incorporates information from all levels of abstraction (i.e., interfaces, operations, and parameters) and aggregates it at the level of operations. For each pair of operations, we determine their similarity on the level of their parent interfaces, on the "native" level of operations, and on the levels of input and output parameters. These individual similarity values are aggregated into a weighted average, which corresponds to the overall similarity of the operations.

Let a denote an operation from the service request and let b denote an operation from the service offer, belonging to an interface u and v respectively. For each pair, we determine:

- The high-level similarity of their respective parent interfaces, $sim_{iface}(a,b)$, given by a comparison of u and v, which is weighted with w_{iface}.
- The center-level, native similarity, $sim_{op}(a,b)$, which is weighted with w_{op}.
- The low-level similarity derived from matching their respective individual input parameters, $sim_{in}(a,b)$, which is weighted with w_{in}.
- The low-level similarity derived from matching their respective individual output parameters, $sim_{out}(a,b)$, which is weighted with w_{out}.

The process is illustrated in Figure 3.3. With $sim_{agg}(a,b)$ denoting the overall similarity of operations a and b, it can formally be expressed as:

$$\begin{aligned} sim_{agg}(a,b) = \quad & sim_{iface}(a,b) * w_{iface} + sim_{op}(a,b) * w_{op} \\ & + sim_{in}(a,b) * w_{in} + sim_{out}(a,b) * w_{out} \end{aligned} \tag{3.1}$$

In order to obtain standardized similarity values, we additionally define:

$$0 \leq w_{iface}, w_{op}, w_{in}, w_{out} \leq 1 \tag{3.2}$$

$$w_{iface} + w_{op} + w_{in} + w_{out} = 1 \tag{3.3}$$

$$0 \leq sim_{iface}, sim_{op}, sim_{in}, sim_{out} \leq 1 \tag{3.4}$$

As a result, the overall similarity is restricted by a lower and upper bound:

$$0 \leq sim_{agg} \leq 1 \tag{3.5}$$

Notably, there are some similarity metrics which cannot be easily normalized to a specific range. In the context of this thesis, this applies to sim_{Resnik} as will be introduced in Equation 3.18. Hence, the constraints from Equation 3.4 and consequently Equation 3.5 cannot always be met.

Once similarities between all pairs of operations in a service request and service offer have been computed, the overall service similarity, sim_{serv}, is derived by finding an optimal matching of operations. I.e., the final matching for a pair of services is conducted between their respective union set of operations, disregarding how the operations are actually organized into interfaces. Formally, let I and J be the sets of operations in a service request R and offer O respectively. Let x_{ij} be a binary variable, indicating whether $i \in I$ has been matched with $j \in J$. Then,

$$sim_{serv}(R,O) = \frac{1}{|I|} * \sum_{i \in I, j \in J} x_{ij} * sim_{agg}(i,j) \tag{3.6}$$

For the example in Figure 3.4, operations (a,b), (c,f), and (e,f) are paired (indicated by bold edges), because they all share a high degree of similarity (indicated by the aggregated similarity values on the edges). The example also outlines a major advantage of matching on the level of operations instead of interfaces: Whilst operation a from interface u is paired with operation b from interface v, operation c – which also belongs to interface u – can still be associated with its best match f, even though the latter belongs to interface y.

In contrast, if interfaces were to be paired, u would be associated with v and x with y. In this case, the overall similarity between the interfaces – and thus, the services – would be low, because their respective operations simply do not match very well. In contrast, if interface u were associated with y and interface x with v, the respective number of operations in the interfaces would mismatch. Again, overall service similarity would be assessed as low. However, as previously outlined, the service offer actually provides

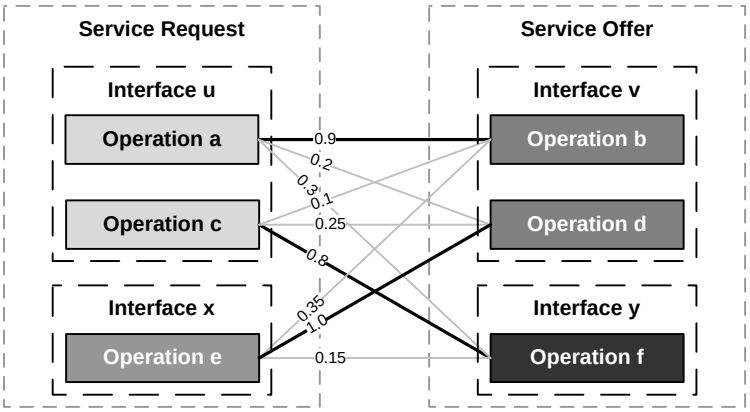


Figure 3.4: Matching of Operations (Example)

most of the functionality that is required by the service request, with the sets of operations matching very well. Thus, matching on the level of interfaces may falsify the service similarity assessment.

Apart from this, our approach also addresses an aspect that has been subject to discussion and controversy in the research community: In the work of Paolucci et al., outputs are taken into consideration for matchmaking first; matching of inputs is only conducted if necessary, i.e., if there are several services with the same DoM after output matching [201]. However, in many other approaches, inputs and outputs are weighted the same, e.g., in [19, 212]. In the work at hand, we allow for different weighting of inputs and outputs without specifying an order in which these parameters are evaluated.

Lastly, the weighting for the merging of similarities can be arbitrarily chosen, thus making our approach very flexible. Notably, by setting certain weights to 0, the respective levels of abstraction can be generally excluded from the matchmaking process. For instance, in some cases, the interface level – or its semantic annotations – might not provide any information that is of use in the matching process. For the evaluation of LOG4SWS.KOM, we have chosen different weightings, which are explained in Section 3.4. In COV4SWS.KOM, the weighting is automatically derived from a training set of services (cp. Section 3.3).

Choice of Similarity Measures

Our primary strategy for similarity assessment in LOG4SWS.KOM is based on DoMs from logic subsumption reasoning. For each pair of components in a SAWSDL service description, their first referenced semantic concepts (as specified by the `modelReference` attribute) are matched against each other. Further model references are not regarded; this is done as it is unclear if a further URI addresses an alternative semantic concept, an additional information or something else, e.g., preconditions and effects as proposed in WSMO-Lite [251]. If logic-based similarity measures cannot be derived, an IR-based fallback strategy is applied (cp. Section 3.2.2).

A set of discrete DoMs has been suggested by Paolucci et al. [201] and applied in similar form in, e.g., [17, 45, 98, 140]. These DoMs are based on subsumption relationships – the basic idea is that if a concept D is more general than a concept C, it subsumes C: $C \sqsubseteq D$. Vice versa, a concept subsumed by another concept is more specific (or a *subconcept*) of the subsuming class (or *superclass*) [11]. We slightly depart from the definition of Paolucci et al. by defining generic types of matches that can be applied to each matching level and type of parameter. Given two arbitrary concepts, A and B, where A is defined in the service request and B is defined in the service offer, the DoM is given by:

$$DoM(A,B) = \begin{cases} exact & if A \equiv B \\ super & if A \sqsubseteq B \\ sub & if A \sqsupseteq B \\ fail & else. \end{cases} \tag{3.7}$$

According to the scheme by Paolucci et al. [201], our DoM of *super* corresponds to DoM of *subsumes* and *plugin* for inputs and outputs respectively; for *sub*, the order is reversed, and it corresponds to *plugin*

and *subsumes* for inputs and outputs. Paolucci et al. define an *exact* match not only if two classes are the same, but also if an output from the request is an *immediate* subclass of an offer's output [201, 234]. As described in Section 3.1.2, this can be traced back to the basic assumption that a class covers all of its subclasses. Notably, the *subclassOf* relationship is not transitive which makes it disputable. Hence, we decided not to make use of *subclassOf* as a fifth DoM in Equation 3.7.

Usually, there is a predetermined order of DoMs – e.g., Paolucci et al. define this as $exact > plugin > subsumes > fail$. Some researchers invert the concepts of *subsumes* and *plugin* and consequently the ranking of more specific and more generic concepts [17, 45, 68, 164]. As it has been discussed in Section 3.1.2, the decision for one of these approaches is based on some basic assumptions what the annotation with a particular semantic concept actually means. Possibly, both order schemes could be wrong for a particular service domain. Considering the respective drawbacks of both schemes, our definition of DoMs can generically be applied to each matching level, and the assignment of numerical equivalents through an OLS estimator efficiently eliminates the need to explicitly define a ranking of DoMs (cp. Section 3.2.3).

Still, these DoMs only provide a discrete scale and thus solely allow a coarse-grained ranking of services. For instance, no further distinction can be made between two "super" matches. Yet, a more sophisticated ranking might be possible, based on semantic or syntactic information that goes beyond these subsumption relations.

Thus, we map the four discrete DoMs onto a continuous numerical scale, ranging from 0 (no similarity at all) to 1 (perfect similarity). This approach allows the combination with other numerical measures and a much finer-grained ranking of services. In detail, on each individual matching level $L \in \{iface, op, in, out\}$, each DoM $D \in \{exact, super, sub, fail\}$ is assigned a numerical equivalent $d_{L;D}$ in the range [0..1]. Formally,

$$d_{L;D} \in [0..1] \quad \forall L \in \{iface, op, in, out\}, D \in \{exact, super, sub, fail\} \tag{3.8}$$

In order to map the discrete subsumption DoM levels to numerical equivalents, two different approaches are possible: In the related work, several authors have proposed the usage of more or less arbitrarily chosen constants [82, 97, 169]. Another approach is to determine an optimal numerical representation by automatic estimation using, e.g., a machine learning approach. Because of its good runtime performance and easy implementation, we decided to make use of an OLS estimator (cp. Section 3.2.3). To allow a comparison of the OLS-based approach, further configurations of the matchmaker with manually assigned numerical values have also been implemented (cp. Section 3.4.1).

In addition to the DoM based on subsumption relation, the path length between semantic concepts in an ontology [216], as utilized by Kaufer and Klusch respectively Plebani and Pernici, is also calculated [119, 212]. The inclusion of minimal path length in the similarity measure implicitly addresses the problem of overly generic or specific annotation of parameters. Because the path length to the ontology root will generally be comparably long, the combined similarity measure is effectively decreased. This resembles an explicit punishment of such generic annotations. It should be noted that the path length is not a logic-based similarity measure, but an ontology-based measure to detect relatedness between semantic concepts [139, 216].

Both measures are merged into one common result, choosing from a number of predefined merging strategies. To combine the DoM level, derived from subsumption reasoning, with the discrete path length into one continuous measure, the following merging strategies are available:

Division: The DoM level from subsumption reasoning, represented as a numerical value, is divided by the path length. I.e., the overall DoM linearly shrinks with increasing path length.

SqrtDivision: The subsumption DoM level is divided by the square-rooted path length. Thus, the overall DoM shrinks slower with increasing path length, compared to a simple division.

SquaredDivision: The DoM level from subsumption reasoning is divided by the squared path length. I.e., the overall DoM shrinks faster with increasing path length, compared to a simple division.

The optimal choice of a merging strategy depends on the charateristics of the underlying ontology. If an ontology is "narrow", i.e., the number of children is relatively small for each concept, the (perceived) similarity of two concepts will probably not diminish too fast with increasing path length, leading to the *SqrtDivision* strategy being the best pick. In contrast, if an ontology is "wide", i.e., the number of children

is high on average for each concept, the similarity of two concepts will diminish very quickly, favoring the *SquaredDivision* strategy. Without further knowledge about the ontology provided, we can assume the *Division* strategy to constitute the best compromise. Thus, it is utilized as default strategy.

Then, formally, let $PL(A,B)$ denote the shortest path between two concepts A and B in an ontology. Furthermore, let L be the level on which the matching of components that points to these concepts is conducted. The overall similarity $cs(A,B)$ between A and B (and thus, the two related service components) is given by:

$$cs(A,B) = \begin{cases} d_{L.exact} & if A \equiv B \\ d_{L.super}/PL(A,B) & if A \sqsubseteq B \\ d_{L.sub}/PL(A,B) & if A \sqsupseteq B \\ d_{L.fail} & else. \end{cases} \tag{3.9}$$

Assigning similarities between pairs of components can be considered a preparatory step in the matching of services. In the next step, the actual matching is to be conducted using a certain matching algorithm.

Selection of Matching Algorithm

Ideally, each component in the service request is to be associated with the best matching component in the service offer, i.e., for all interfaces, operations, input and output parameters from a request, the best matching equivalent from a service offer should be found and assigned. In the seminal work of Paolucci et al., a greedy approach has been proposed, which has been picked up by many authors [201]. As stated in Section 3.1.3, Bellur et al. have shown that the greedy approach might lead to suboptimal matching results [17].

In LOG4SWS.KOM, we have chosen to implement a bipartite matching algorithm to solve this assignment problem. We follow the example by Plebani and Pernici and also divert from the concept of global DoM [212]. I.e., we maximize the sum of edge weights (corresponding to the similarity of matched components) instead of maximizing the minimal edge weight. As a major advantage, this allows for the application of standard bipartite matching algorithms, most prominently the Hungarian algorithm, and corresponding available implementations.

Single matching results are combined by calculating the average similarity value of all pairwise matchings. This matching result reflects the overall similarity of two sets of components (and thus, ultimately, services) and can loosely be perceived as a measure of effort that would be needed to adapt a service offer to fit a certain request. Here, a high similarity value implies little adaptation effort and vice versa. An approach which guarantees a maximum global DoM instead of an average has been presented by Bellur et al. [17].

A remaining issue concerns the treatment of differing parameter cardinalities. In the case of LOG4SWS.KOM, we have chosen to generally allow for the matching of component sets, regardless of their respective cardinality. This concerns both operations and message parameters, i.e., inputs and outputs. Thus, we do not require a service to offer a corresponding operation or output for each such component in the service request. In line, we do not require a service offer's inputs to be identical to or a subset of the service request's inputs. As the original Hungarian algorithm can only be applied to square matrices, i.e., two sets of identical cardinality, an implementation of the extension for rectangular matrices proposed by Bourgeois and Lassalle has been used [35, 194].

In order to compute the similarity of two matched sets of components with differing cardinalities, a decisive cardinality needs to be determined. We make use the following strategy:

- Generally, the cardinality of the set associated with the service request is decisive. I.e., if an offer lacks requested operations or outputs, its overall similarity decreases.
- For inputs, the cardinality of the set associated with the service offer is decisive. I.e., if an offer requires more inputs than the request provides, its overall similarity decreases.

In this section, the basic version of LOG4SWS.KOM has been introduced, which contains the fundamental concepts applied. However, there are two enhancements that play an important role and will be

considered in separate sections. In Section 3.2.2, we present the fallback strategy applied if semantic annotations are missing or cannot be processed; in addition, we also introduce a caching mechanism applied in order to speed up the overall matchmaking process. In Section 3.2.3 we introduce an OLS-based approach to determine the weights needed to compute the overall similarity value in Equation 3.1. Incorporating the defined matching levels, similarity measures, and matching algorithm, Section 3.2.4 presents the implementation of LOG4SWS.KOM in terms of its main algorithms.

3.2.2 Fallback Strategy and Caching

As it was mentioned before, the number of semantically described Web services is still quite low. Actually, even SAWSDL-TC, the current scientific standard test data collection for SAWSDL-based service discovery (cp. Appendix B.2.3), offers semantic annotations only on the message parameter level. In fact, it is easier to generate semantic annotations on this level, as Web service descriptions are usually generated from source code, e.g., Java code, where data types are already described [212]. For other elements, like interfaces and operations in SAWSDL, such a description is often missing. Furthermore, ontologies which could be used in order to annotate interfaces and operations are still not established on a broad scale [240]. This can be, e.g., traced back to the fact that the formal definition of knowledge in ontologies is often a time-consuming and costly process [103, 235]. Even if a suitable ontology is available, the annotation of Web service descriptions with semantic references is still a cumbersome and considerable task [107, 174]. Hence, it seems unrealistic to assume that SWS descriptions are necessarily correct and complete [180].

If semantic descriptions of certain elements are missing, it might be helpful to make use of the remaining information which describe the individual components: Even though this thesis focuses on the semantic description of service components, we have also implemented a fallback strategy, which can be applied if semantic annotations for a certain element are missing. This strategy may also be utilized if an ontology fails to load for some reason, rendering reasoning about a referenced semantic concept impossible.

The fallback strategy allows to compute the similarity between any pair of names, including the names of components (if semantic annotations are completely missing) or the names of two semantic concepts (if an ontology fails to load). The similarity measure is computed using the *WordNet* ontology [186]. Again, the similarity is a numerical value between 0 and 1. The fallback strategy can also be applied to different levels, e.g., if semantic annotations are missing on the operation level, as long as the components on these levels possess names.

All names undergo the following processing:

1. Names are split into individual tokens, based on commonly used separators, such as underscore (“_”) and dash (“-”), or the *camelCase* notation.
2. If these tokens do not form a word according to the WordNet ontology, they are stemmed using a Porter stemmer [213], and the resulting root form is again checked against the WordNet ontology [186].
3. If this check fails again, the token is scanned for meaningful substrings in a recursive manner.

This way, names such as “get_flight_price”, “getFlightPrice”, and “getflightprice” can be effectively split into individual words.

Each set of words constitutes a partition for a bipartite graph. The edge weight corresponds to the minimal inverse distance of a pair of words in WordNet. Consecutively, bipartite graph matching is employed, with the average edge weights in the matching yielding the similarity of the two names and, thus, two service components. If the count of words differs in the two partitions, the larger set of words determines the overall cardinality in computing the average.

In order to improve the performance of LOG4SWS.KOM in terms of query response time, we utilize different caches, which may be populated both at registration and query time. Each cache is implemented as a hash table, storing one or more data items for each unique key. In detail, the following caches exist:

- The *subsumption cache* stores the type of subsumption match (exact, super, etc.) and path length between two semantic concepts. The concepts’ names conjointly constitute the caching key. Important to note, the cache does not store the computed similarity value. It thus allows to freely adapt

the numerical DoM equivalents and merging strategy without raising the need to repopulate the cache.

- The *word split cache* holds all words that can be derived from a name by splitting it, using the aforementioned process that is applied in the fallback strategy. In line, the initial name forms the cache's key.
- The *word distance cache* stores the distance between two words according to the WordNet ontology. Again, the pair of words conjointly constitutes the key.

In practical application, preliminary caching should be employed at service publication time by matching the new service offer against all service offers that have previously been registered. Subsequently, at query time, if a service request contains already cached semantic concept and syntactic descriptions – which becomes more likely with an increasing number of services having been previously matched – the similarity values can be retrieved very fast from the cache. This may significantly improve query response times. Through utilization of caches, costly computations need only to be conducted if the corresponding data item is not part of one of the service offers and included in a service request for the first time. The item will then be automatically stored in the cache and available for upcoming requests. Similar approaches have been proposed, e.g., in [145, 234].

3.2.3 Deriving Numerical Equivalents of Discrete DoMs Using OLS

As previously outlined, the assignment of continuous numerical equivalents to discrete matching levels is an ambiguous process. In order to derive more fine-grained similarities between semantic concepts and allow for the merging with other numeric measures, such mapping is inevitable, however.

LOG4SWS.KOM applies an OLS estimator for the determination of optimal numerical DoM equivalents. We assume that the weighted linear combination of the different DoM's frequencies predicts the similarity of two operations, and thus, ultimately, two services. The process is based on the notion that a dependent variable $y^{a/b}$ – corresponding to the similarity of two operations a and b – can be derived through the linear combination of a set of independent variables $x_{L;D}^{a/b}$, corresponding to the frequency of a certain DoM D on a certain matching level L when matching a and b.

Under the assumption that the sum of squared residuals is to be minimized, OLS comprises an optimal, unbiased estimator [121, 257]. Residual, in that context, refers to the absolute difference between the actual and the estimated similarity of a service offer with respect to a service request. As a major advantage, implementations of OLS are available for multiple platforms and generally perform very well, involving little computational cost [121].

The OLS estimation is independently conducted for each matching level, i.e., the numerical weights differ for input and output parameters, operations, and interfaces. As training data, a set of services is required along with a predefined similarity rating. A subset of a test collection, such as SAWSDL-TC (cp. Appendix B.2.3), fulfills this requirement, because it provides an example scenario consisting of queries and associated relevance sets.

Test collections often utilize a coarse-grained, graded relevance scale to express services' similarity. Such scale is *discrete* in nature, whilst – at least in case of LOG4SWS.KOM and COV4SWS.KOM – similarity is expressed on a *continuous* scale. Accordingly, the relevance grades have to be consistently mapped to numerical values. With respect to SAWSDL-TC, for instance, the two binary relevance grades of *not relevant* and *relevant* can be mapped to 0 and 1 respectively. This scheme could be adapted for more extensive graded relevance schemes like those proposed by Tsetsos et al., Tran et al., or Küster and König-Ries [152, 154, 245, 246]. Vice versa, continuous similarity values can be mapped back into discrete relevance grades by defining ranges of correspondence. For instance, a similarity value threshold of 0.5 or more might indicate a relevant operation or service according to a binary scale.

In the training phase, LOG4SWS.KOM matches all pairs of operations in all service requests and offers. For each pair and each matching level, it stores the types of subsumption matches in the matched components along with the path length, and retrieves the predefined similarity of the two operations. If such figure is unavailable at the operation level, the corresponding value on the higher levels of interfaces or services will be used. The similarities constitute the vector of predictors (y) for the OLS process, with each row corresponding to a pair of operations. The individual design matrices (X_{iface}, X_{op}, X_{in}, and X_{out})

are derived in the following manner: Each pair of operations yields one row with four entries, where each entry ($x^{a/b}_{L;D}$, that is, in the example of two operations a and b) corresponds to the frequency of a certain type of DoM D with respect to all matched components on a certain level L.

More precisely, the frequency count is incremented by 1 for an exact and fail match between two components. For super and sub matches, it is incremented by 1 merged with the path length. As outlined in the previous section, some merging strategy will have to be chosen – by default, this is *division*, i.e., 1 is divided by the path length. The row is finally divided by the total number of matched components on the current level.

Assume, for instance, that a pair of operations, a and b, is matched. If each operation contains two inputs, there is a total of two input matches. Let us further assume that there is an exact DoM for the first pair of matched inputs and a super match with a path length of 2 for the second pair of matched inputs. This results in the following row in the input level's design matrix (where the first entry corresponds to an exact DoM, the second to super, the third to sub, and the fourth to fail):

$$x^{a/b}_{in} = \begin{pmatrix} 1/2 & \frac{1}{2}/2 & 0 & 0 \end{pmatrix} \tag{3.10}$$

Under the assumption that the operations a and b are relevant with respect to a binary scale (which, again, translates into a similarity of 0 for non-relevant and 1 for relevant pairs of operations), the corresponding entry in the vector of predictors is:

$$y^{a/b} = \begin{pmatrix} 1 \end{pmatrix} \tag{3.11}$$

An exemplary design matrix (for the level of inputs) and vector of predictors are depicted in the following, with the first row containing the data for a pair of operations a and b (which is mutually relevant according to the binary scale shown in Equation 3.13), and the second row containing data for a pair a and c (which is not mutually relevant according to the binary scale):

$$X_{in} = \begin{pmatrix} x^{a/b}_{in;exact} & x^{a/b}_{in;super} & x^{a/b}_{in;sub} & x^{a/b}_{in;fail} \\ x^{a/c}_{in;exact} & x^{a/c}_{in;super} & x^{a/c}_{in;sub} & x^{a/c}_{in;fail} \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix} = \begin{pmatrix} 0.5 & 0.25 & 0 & 0 \\ 0 & 0 & 0.\overline{3} & 0.5 \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix} \tag{3.12}$$

$$y = \begin{pmatrix} y^{a/b} \\ y^{a/c} \\ \vdots \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ \vdots \end{pmatrix} \tag{3.13}$$

Given a design matrix and vector of predictors, the standard OLS estimator can be applied in the following manner, with $L \in \{iface, op, in, out\}$ denoting the matching level:

$$\hat{\beta}_L = (X'_L X_L)^{-1} X'_L y \tag{3.14}$$

$\hat{\beta}_L$ corresponds to the optimal estimate of numerical weights, whereas X_L denotes the design matrix and y the corresponding vector of predictors on a certain level L.

In order to derive actual numerical DoM equivalents, we further process the vector. In detail, all entries of the vector are mapped to the range $[0..1]$. For that matter, the minimum value in the vector is added to all entries. Then, all entries are divided by the new maximum value. The underlying idea is that – even though there might be a negative equivalent of a DoM – these negative values could lead to overall negative similarity values sim_{agg} and sim_{serv}, which are not intended.

For instance, assume that the OLS estimator outputs the following estimate for the level of inputs:

$$\hat{\beta}_{in} = \begin{pmatrix} 1.1 & 0.7 & 0.3 & -0.1 \end{pmatrix} \tag{3.15}$$

The minimum value, $|-0.1|$, is consecutively added to each entry, yielding

$$\hat{\beta}^{+}_{in} = \begin{pmatrix} 1.2 & 0.8 & 0.4 & 0 \end{pmatrix} \tag{3.16}$$

Dividing all entries by the new maximum, 1.2, results in the final vector of numerical DoM equivalents d, namely

$$d_{in} = \begin{pmatrix} 1.0 & 0.\overline{6} & 0.\overline{3} & 0 \end{pmatrix} \tag{3.17}$$

Based on aforementioned order of entries, the numerical equivalents for the individual DoM types on the level of inputs then correspond to:

$$\begin{aligned} d_{in;exact} &= 1.0 \\ d_{in;super} &= 0.\overline{6} \\ d_{in;sub} &= 0.\overline{3} \\ d_{in;fail} &= 0 \end{aligned}$$

For the purpose of cross-validation (cp. Appendix B.4), the design matrices and vectors of predictors are adapted. Specifically, all rows are removed from both the matrix and vector that either refer to the currently validated service request or do not refer to a service offer in the previously determined set of relevant service offers. This process ensures that no information which derives from the currently validated service request is used in the OLS regression.

3.2.4 Implementation

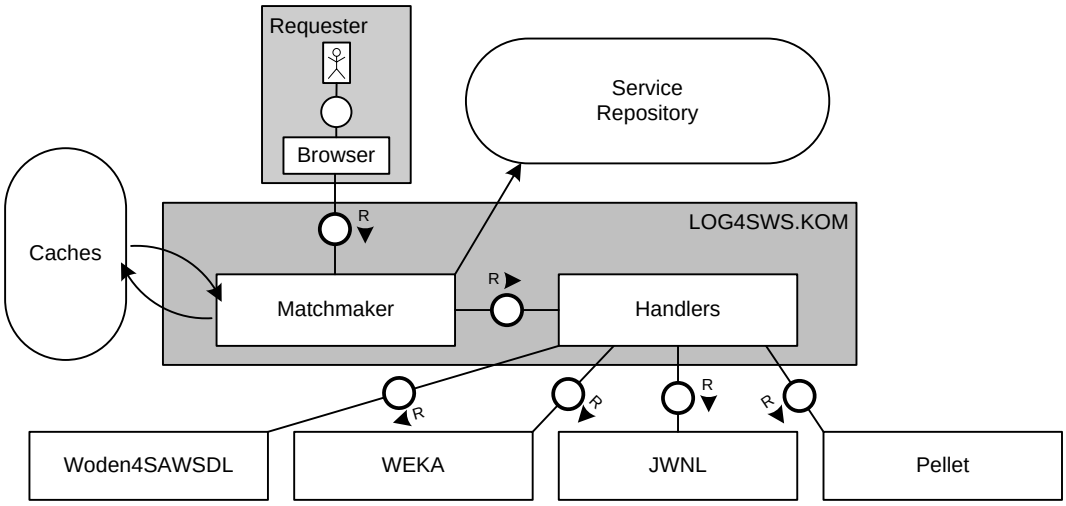


Figure 3.5: Implementation Overview for LOG4SWS.KOM

In the following, the implementation of LOG4SWS.KOM in terms of its main components and algorithms is presented. Figure 3.5 shows how LOG4SWS.KOM interacts with the service repository and the employed Application Programming Interfaces (APIs):

Pellet is an OWL DL Reasoner used for logic-based matching [230].

Weka provides a number of machine learning algorithms as well as preprocessing facilities for syntax-based similarity measurement [256].

JWNL is a Java library for WordNet [186].

Woden4SAWSDL facilitates the processing of SAWSDL-based service descriptions in Java.

Notably, Figure 3.5 provides a matchmaker perspective, i.e., details of the service repository and the caches are omitted. As it can be seen, LOG4SWS.KOM is essentially made up from two components – the matchmaker itself as well as a number of handlers for the different APIs. These handlers are

Listing 3.1: Function matchServices

```
Input:      regService, offService: Services to be matched                                                    1
Output:     overallSim: Similarity between the both services                                                  2
Variables:  reqOp[i], offOp[j]: Operations                                                                    3
            assignment: One-to-one assignment of operations                                                   4
            simIface, simOp, simIn, simOut: Similarity values of component pairs                              5
            wIface, wOp, wIn, wOut: Weights                                                                   6
            sim: Matrix of aggregated similarity values                                                       7
                                                                                                              8
function matchServices(reqService, offService) {                                                             9
   for each (reqOp[i] in reqService.operations)                                                               10
      for each (offOp[j] in offService.operations)                                                            11
         simIface(i,j) = calculateSimilarity(reqOp[i].parentInterface, offOp[j].parentInterface)              12
         simOp(i,j) = calculateSimilarity(reqOp[i], offOp[j])                                                 13
         simIn(i,j) = matchParameters(reqOp[i].inputs, offOp[j].inputs)                                       14
         simOut(i,j) = matchParameters(reqOp[i].outputs, offOp[j].outputs)                                    15
         sim(i,j) = wIface * simIface(i,j) + wOp * simOp(i,j) + wIn * simIn(i,j) + wOut * simOut(i,j)         16
      end for each                                                                                            17
   end for each                                                                                               18
   assignment = MatchingAlgorithm.match(sim)                                                                  19
   overallSim = EvaluationStrategy.evaluate(sim, assignment)                                                  20
   return overallSim                                                                                          21
}                                                                                                             22
```

Listing 3.2: Function calculateSimilarity (LOG4SWS.KOM)

```
Input:      regComponent, offComponent: Service components of the same type (interfaces or operations)       1
Output:     calculateSubsumptionSimilarity, calculateNameSimilarity: Similarity between the both components  2
Variables:  reqComponent.mref, offComponent.mref: References to semantic concepts                             3
                                                                                                              4
function calculateSimilarity(reqComponent, offComponent) {                                                   5
   if reqComponent.mref != null and offComponent.mref != null                                                 6
      if ontologiesAvailable(regComponent.mref, offComponent.mref)                                            7
         return calculateSubsumptionSimilarity(reqComponent.mref, offComponent.mref)                          8
      else                                                                                                    9
         return calculateNameSimilarity(regComponent.mref, offComponent.mref)                                10
      end if                                                                                                  11
   else                                                                                                       12
      return calculateNameSimilarity(regComponent.name, offComponent.name)                                    13
   end if                                                                                                     14
}                                                                                                             15
```

applied in order to decouple the API methods from the matchmaking facilities and allow to replace single components. It should be mentioned that LOG4SWS.KOM is not directly applied to SAWSDL files, as the deserialization of these files takes comparably much time. Instead, service offers are deserialized into the ATSM format presented in Appendix A.1.3. As ATSM contains both a SAWSDL and OWL-S mapping, it is furthermore possible to apply the matchmakers to OWL-S, too. However, for evaluation purposes, we restrict ourselves to SAWSDL.

The main algorithm applied in LOG4SWS.KOM is `matchServices(request, offer)` which matches two services *reqService* (the service request) and *offService* (the service offer) to determine the similarity between the particular operations. In this context, services are represented as tree structures in ATSM (cp. Appendix A.1.3). For reasons of simplicity, we assume that all operations are directly associated with the respective service, omitting the intermediate level of interfaces. As it can be seen from Listing 3.1, for each combination of operations from service request and offer, the four distinct similarity values as presented in Section 3.2.1 are computed. Afterwards, an optimal matching is assigned (Line 19) and the overall similarity for the two services is calculated (Line 20). Both aggregate the optimal operation matching and the operations pairs' similarity values into a common service similarity figure.

The native similarity values for interface and operation level are calculated using `calculateSimilarity` as depicted in Listing 3.2. In this method, the methods `calculateSubsumptionSimilarity` (Line 8) and `calculateNameSimilarity` (Lines 10 and 13) are of particular importance. These functions are depicted in Listings 3.4 and 3.5.

Listing 3.3: Function matchParameters

```
1  Input:      reqOperations.parameters, offOperation.parameters: Parameters to be matched
2  Output:     overallSim: Similarity between the both parameter sets
3  Variables:  reqPara[i], offPara[j]: Parameters
4              assignment: One-to-one assignment of parameters
5
6  function matchParameters(reqOperation.parameters, offOperation.parameters) {
7     for each (reqPara[i] in reqOperation.parameters)
8        for each (offPara[j] in offOperation.parameters)
9           sim(i,j) = calculateSimilarity(reqPara[i], offPara[j])
10       end for each
11    end for each
12    assignment = MatchingAlgorithm.match(sim)
13    overallSim = EvaluationStrategy.evaluate(sim, assignment)
14    return overallSim
15 }
```

Listing 3.4: Function calculateSubsumptionSimilarity

```
1  Input:      A, B: Semantic concepts from service request and service offer respectively
2              matLevel: The level of abstraction (iface, op, etc.) on which the concepts are matched
3  Output:     sim: Similarity value of the concepts A and B
4  Variables:  subResult: A complex subsumption matching result with matching type and path length
           between two concepts
5              numEquiv: The numerical equivalent of a discrete DoM of the current service abstraction level
6              matLevel: The current service abstraction level
7
8  function calculateSubsumptionSimilarity(A, B, matLevel) {
9     if cache.contains(A, B)
10       subResult = cache.get(A, B)
11    else
12       if A is equivalent to B
13          subResult.domLevel = EXACT
14          subResult.pathLength = 1
15       else if B subsumes A
16          subResult.domLevel = SUPER
17          subResult.pathLength = getPathLength(A, B)
18       else if A subsumes B
19          subResult.domLevel = SUB
20          subResult.pathLength = getPathLength(A, B)
21       else
22          subResult.domLevel = FAIL
23          subResult.pathLength = 1
24       end if
25    end if
26    cache.store(A, B, subResult)
27    numEquiv = getNumericalEquivalent(subResult.domLevel, matLevel)
28    sim = MergingStrategy.merge(numEquiv, subResult.pathLength)
29    return sim
30 }
```

The similarity of message parameters is calculated analog to that of the other service components. However, there is one particular difference: As can be seen from Listing 3.3, `matchParameters` executes another optimal matching for parameters (Line 12). Afterwards, the overall similarity for the two sets of parameters is calculated (Line 13) and returned (Line 14).

As the name implies, `calculateSubsumptionSimilarity` computes the subsumption reasoning-based DoM between two semantic concepts (cp. Listing 3.4). If the DoM has not been stored in the corresponding cache (Line 9), it is computed at runtime (Lines 11–25). The actual similarity value between the concepts is made up from the DoM and the path length as presented in Section 3.2.1. Afterwards, the new similarity result is stored in the cache (Line 26), and the numerical equivalent of the subsumption DoM for the corresponding matching level and current service abstraction level is determined (Line 27). Finally, the similarity value is calculated by merging the numerical equivalent with the path length (Line 28) using a predefined merging strategy and returned (Line 29).

If `calculateSubsumptionSimilarity` cannot be applied, the name similarity of two service components, which might be strings referencing semantic concepts or the name attributes of the components, is calculated using `calculateNameSimilarity` (cp. Listing 3.5). Here, the names are first split into sets of individual words (Lines 7 and 8). The corresponding function, `splitIntoWords`, uses the word split

Listing 3.5: Function calculateNameSimilarity

```
Input:      reqName, offName: Name from the service request and service offer respectively
Output:     sim: Similarity value of the two names
Variables:  reqWords, offWords: Sets of terms derived from the names
            sim(i,j): Similarity value of the terms

function calculateNameSimilarity(reqName, offName) {
   reqWords = splitIntoWords(reqName)
   offWords = splitIntoWords(offName)
   for each (reqWords[i] in reqWords)
      for each (offWords[j] in offWords)
         sim(i,j) = 1 / getWordNetDistance(reqWords[i], offWords[j])
      end for each
   end for each
   assignment = MatchingAlgorithm.match(sim)
   overallSim = EvaluationStrategy.evaluate(sim, assignment)
   return overallSim
}
```

cache in order to look up names that have been previously split, respectively stores every new name along with its list of words automatically . For each combination of words from these sets, the similarity based on the WordNet distance is calculated (Lines 9–13). Again, the function `getWordNetDistance` will access the word distance cache to look up known pairs and store new entries. After distances have been assigned for all pairs of terms, an optimal matching is computed (Line 14) and the overall similarity for the two names is calculated (Line 15).

3.3 COV4SWS.KOM: Matchmaking Based on Semantic Relatedness for SAWSDL

Even though LOG4SWS.KOM enhances current matchmaking approaches for SAWSDL by new ideas and concepts, it is still based on subsumption reasoning and uses discrete and thus relatively coarse DoMs. In contrast, COV4SWS.KOM applies an alternative measurement of similarities between semantically annotated objects: Similarity measurement is based on the idea that the similarity of semantic concepts differs depending on how related they are with respect to each other and with respect to their placement in an ontology. This way, COV4SWS.KOM incorporates more fine-grained semantic-based relationships than the usually applied subsumption matching-based approaches.

As a linear regression is no further needed in order to derive numerical equivalents of DoMs, it is possible to deploy the OLS estimator on another set of variables – for COV4SWS.KOM, an OLS estimator is used in order to determine to which degree the different service abstraction levels should be incorporated into the overall matchmaking results. The basic assumption is that different service abstraction levels should be given a weight according to their impact on the overall result.

Apart from these aspects, COV4SWS.KOM is making use of the same mechanisms presented for LOG4SWS.KOM, especially regarding the fallback strategy as well as the caching mechanisms (cp. Section 3.2.2), the combination of similarity values from different service abstraction levels (however, weights are automatically derived), and the usage of the Hungarian algorithm for the actual matching of service components (cp. Section 3.2.1).

The remaining part of this chapter is structured as follows: In the next section, we present the foundations of semantic relatedness and the applied similarity metrics. Afterwards, the derivation of weightings for different service abstraction levels is presented. In Section 3.3.3, the actual implementation is presented.

3.3.1 Semantic Relatedness

In COV4SWS.KOM, the assignment of similarity values does not employ the traditional strategy of subsumption reasoning (based on discrete levels such as exact, plugin, etc.) between concepts, as suggested by Paolucci et al. [201] and applied in similar form in, e.g., [17, 45, 140, 238]. Subsumption-based matchmaking suffers from a number of drawbacks. For one, it may reward the annotation with overly generic concepts and thus lead to suboptimal matchmaking results. This makes it necessary to use

further aspects to penalize overly generic annotations as it has been done in LOG4SWS.KOM. Second, subsumption-based DoMs are quite coarse-grained and do not incorporate additional information available from the ontology structure, such as the distance between two concepts or the degree of increasing specialization between levels. Third, the combination with usually numerical similarity values from IR is generally not easy to achieve. Last but not least, subsumption-based DoMs rely on a ranking, which can to some degree be quite arbitrarily.

Hence, COV4SWS.KOM computes the similarity between two semantic concepts in an ontology based on the relatedness of these concepts. The assignment of semantic relatedness of concepts in an ontology or taxonomy is a well-known problem from computational linguistics and artificial intelligence, resulting in different similarity measures, e.g., [166, 167, 219]. In contrast to the logic-based subsumption matching presented in LOG4SWS.KOM, non-logic-based semantic relatedness possesses a certain degree of uncertainty, as it is the case with IR-based similarity measures – semantically related objects might still not be similar and may lead to both false positives and false negatives [40]. Nevertheless, such approaches provide a multitude of well-explored methodologies. As it was mentioned in the previous sections, there is no generally accepted method to annotate Web services with semantics (especially for SAWSDL). This applies both to the way how semantic concepts are related to each other in an ontology and what the semantic annotation of a service component actually means. Thus, methods from the field of semantic relatedness might provide significant contribution to SWS matchmaking – both as a complement as well as a substitute to logic-based similarity measures.

Because of the large number of different approaches (an extensive survey of semantic relatedness measures is provided by Budanitsky and Hirst [40]), we decided to implement a selection of similarity measures from this field in COV4SWS.KOM in order to show the applicability of such measures. Most of the existing approaches make use of a graph representation of an ontology, where the graph nodes represent the semantic concepts and the links/edges between the nodes represent relationships between the concepts. One way to compute the relatedness between nodes in such a graph would be the measurement of the distance between the graph nodes as applied in, e.g., Uddi Registry By Example (URBE) [212]. Such approaches can usually be traced back to the seminal work by Rada et al. and are named *edge counting*-based [216]. As Resnik states – even though this approach is very intuitive – the path length is often not sufficient as links in a taxonomy or ontology do not necessarily represent the same distance [219]. Resnik provides another measure. Here, the similarity between two semantic concepts is defined as the amount of information they share. In turn, this amount is based on a common ancestor in an ontology. As a result, equality between semantic concepts does not necessarily lead to the highest similarity and for subsumption relationships, the similarity is directly reliant on the most informative ancestor class. Approaches based upon the work of Resnik are named *information theory*-based.

To realize this approach, all concepts in an ontology are augmented with values which indicate the *probability* that an instance of a concept is encountered. The actual similarity of two concepts A and B is based on the probability p assigned to their most informative ancestor $anc(A,B)$. Probabilities are monotonically nondecreasing if moving up the taxonomy; if an ontology possesses a unique top node, its probability is 1. This can be traced back to the fact that classes inherit the probability values of their subclasses. After the probability has been determined, it is possible to derive the *information content* of $p(anc(A,B))$ which is defined as negative log likelihood. Formally, according to Resnik, the similarity between the semantic concepts A and B is:

$$sim_{Resnik}(A,B) = -\log p(anc(A,B)) \tag{3.18}$$

Resnik's metric is not normalized to a specific range, i.e.

$$0 \leq sim_{Resnik}(A,B) \leq \infty \tag{3.19}$$

Hence, Equation 3.4 and 3.5 do not necessarily apply if this metric is applied.

Another information theory-based similarity measure has been presented by Lin [167]. Here, the author makes use of three basic "intuitions":

(i) The similarity between two objects A and B is related to their *commonality*. The more commonality they share, the more similar they are.

(ii) The similarity between two objects A and B is related to the differences between them. The more differences they have, the less similar they are.

(iii) The maximum similarity between A and B is reached when A and B are identical. In this case, it does not matter how much commonality they share.

Based on these intuitions, Lin derives a couple of assumptions which ultimately lead to the definition of a similarity theorem, which essentially defines that the similarity between A and B can be measured by the ration between the amount of information needed to state their commonality and the information needed to fully describe what they are. Applied to a taxonomy, the similarity can be measured as follows:

$$sim_{Lin}(A,B) = \frac{2 * \log p(anc(A,B))}{\log p(A) + \log p(B)} \tag{3.20}$$

As it can be seen from Equation 3.20, Lin's similarity measure also makes use of the probability as defined by Resnik, but normalizes the combination of information content [166]:

$$0 \leq sim_{Lin}(A,B) \leq 1 \tag{3.21}$$

Probability values p are derived in the same way as proposed for sim_{Resnik}, which leads to a similar problem in both approaches. In the original work by Resnik, the similarity measurement is applied to WordNet. Hence, it is possible to derive the probability values from the occurrences of words in an English language corpus [219]. In the case of SWS matchmaking, this is more difficult to achieve as a corresponding corpus is usually missing, or not as extensive as a term corpus that might be applied to derive probabilities in WordNet. Even though we used SAWSDL-TC for the determination of probabilities, it needs to be noted that SAWSDL-TC is comparably small (7115 semantic concepts referenced from 894 Web services) in comparison to the *Brown University Standard Corpus of Present-Day American English* with its more than one million words [83], which has been applied by Resnik.

Hence, apart from deriving the p-values from SAWSDL-TC, we propose a second way to detect probabilities: We apply a probability of 1 to the topmost semantic concept found in an ontology (and equivalent classes, if any exist) as it is done by Resnik. The probability for the direct descendants of a class is then determined by counting its siblings in an ontology, i.e., all classes that share a direct ancestor get the same probability value. As all concepts in an ontology cover a certain amount of the probability value assigned to the topmost semantic concept and vice versa, we name the probability Mutual Coverage (MC). We are well aware, that this way to determine probabilities is to some degree arbitrarily, especially the "equal distribution" of probabilities. However, MC adheres to Resnik's basic assumption that probability values are monotonically nondecreasing, as classes inherit the probability values of their subclasses. Furthermore, using MC, it is possible to define a p-value even for those semantic concepts that are not regarded in the annotations of a corpus. Finally, MC allows to derive a further similarity metric as will be presented in the following. An example is provided in Figure 3.6, which is an augmented variant of Figure 3.1. For each class (i.e., semantic concept), the probability (i.e., MC) is given – *PrintedMaterial* is the topmost semantic concept in this example. *Publication*, for instance, has a MC of 1.0 with *PrintedMaterial* as it is the only subconcept. In line, *Magazine* has a MC of 1/6 with *PrintedMaterial*, because it is 1 out of 3 subclasses of *Serial-Publications*, which in turn has a MC of 1/2.

In COV4SWS.KOM, we investigate the applicability of information theory-based similarity measures in SWS matchmaking. Therefore, we apply sim_{Resnik} and sim_{Lin} instead of subsumption-based similarity. As a reference value, we also determine similarity by measuring the shortest path length between two semantic concepts in an ontology (taxonomy), sim_{PL}, as proposed by Rada et al. [216].

Furthermore, we derive the similarity directly from the MC – here, we make use of two subconcept relationships from the field of DL, i.e., *disjointness* and *covering* (cp. Appendix A.3.3). Given a parent concept, we assume that its direct subconcepts (or children) are disjoint and the set union precisely corresponds to the parent concept, i.e., the latter is covered by its subconcepts [34]. The same requisite is made by Paolucci et al. [201]. So, without further knowledge provided, each subconcept can be assumed to cover (or represent) a portion of the superconcept. This similarity metric, sim_{MC} is based on the assumption that two concepts are similar to each other if their superconcepts are the same [75]. The actual similarity is determined based on the kind of relationship the two concepts share with themselves and their common ancestor (i.e., superconcept).

It needs to be mentioned that we assume the definition of disjointness and covering for semantic concepts, even if they are not explicitly defined in an ontology. For SAWSDL-TC, this is a valid assumption

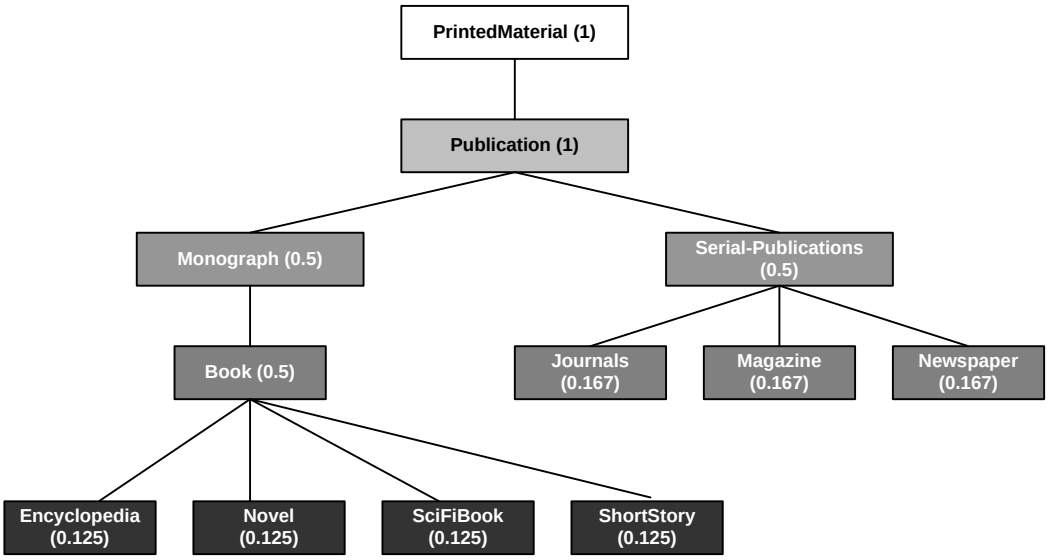


Figure 3.6: Computation of Coverage Between Concepts (Example)

as every service element is annotated with at most one reference to a semantic concept, i.e., there is no individual belonging to more than one class. However, this assumption differs from the general definition of disjointness in description logics, where two classes are disjoint if the set of common individuals *must* be empty. In the data set at hand, the set of common individuals *is* actually empty, but not because of the ontological restrictions, but because of service design decisions. Wang has stated that only a small fraction of ontologies include disjointness axioms [255]. This could be traced back to the fact that users might assume that classes are partitions, if building subsumption hierarchies [252], i.e., disjointness might have not been modeled even though intended by ontology designers. From a DL point of view, Borgida and Brachman state that subclasses/subconcepts are disjoint from each other in many cases [34].

Regarding the abovementioned MC, the probability of a concept C is $p(C) = sim_{MC}(owl{:}Thing, C)$, as p determines the coverage between the top node in an OWL DL ontology, i.e., `owl:Thing`, and the concept at hand (except for concepts equivalent to `owl:Thing`). For the according similarity metric, this model needs to be enhanced in order to compute similarity values for non-subsumption relationships between two concepts:

Again, let A and B denote two semantic concepts. Furthermore, let $dsc(A) = \{C_1, \ldots, C_n\}$ denote A's direct subconcepts, or children. Specifically, children or those descendants of a concept that are precisely connected to it through precisely one *is-a* link. Let $A \sqsubseteq B$ denote that A is a (direct or indirect) subclass and thus specialization of B. Formally, the set union of all children of A is equivalent to A itself:

$$\bigcup_{i=1}^{n} C_i = C_1 \cup \cdots \cup C_n \equiv A \tag{3.22}$$

The coverage between a concept and each of its children is given by the inverse of the number of children:

$$MC(A, C_i) = \frac{1.0}{|dsc(A)|} \quad \forall C_i \in dsc(A) \tag{3.23}$$

We distinguish three types of principal relations between semantic concepts, for each of which the MC is defined in a specific manner.

Equivalence: Two semantic concepts are equivalent or identical. This corresponds to an *exact* match as defined in LOG4SWS.KOM. In case of equivalence, the similarity and thus MC between concepts is 1.0, because equivalent concepts intuitively fully cover each other. Formally,

$$sim_{MC}^{eq}(A, B) = \begin{cases} 1.0 & if A \equiv B \\ 0.0 & else \end{cases} \tag{3.24}$$

Subsumption: Two semantic concepts are related through a linear line of ancestry, i.e., one concept is a direct or indirect subclass and thus specialization of another. Differently stated, the second concept *subsumes* the first. This corresponds to a *super* or *sub* in LOG4SWS.KOM. For the direct children of concept A, the MC is defined as

$$sim_{MC}^{dsc}(A,B) = \begin{cases} \frac{1.0}{|dsc(A)|} & if\, B \in dsc(A) \\ 0.0 & else \end{cases} \tag{3.25}$$

In case of indirect relations between concepts, the MC may be recursively determined. Let $L(A,B) = \{L_1, \ldots, L_n\}$ denote a line of ancestry between A and B. L_1 corresponds to the most generic concept, A, L_n is the most specific concept, B. L_2 through L_{n-1} denote the concepts that form the intermediate line of ancestry with increasing specialization between A and B. Formally, it holds that

$$L_{i+1} \in dsc(L_i) \quad \forall L_i \in la(A,B) \wedge 1 \leq i \leq |la(A,B)| \tag{3.26}$$

L_2 relates to A through a simple subsumption relation, resulting in a certain similarity value. Consecutively, L_3 may cover A to that extent at the best, because in turn, it covers a portion of L_2; the MC between L_3 and A is consecutively given by the product of $sim_{MC}(A, L_2)$ and $sim_{MC}(L_2, L_3)$. The same argument holds for each further descendant of A. In line, the overall MC for a line of ancestry L is computed as

$$sim_{MC}(L) = \prod_{i=1}^{n-1} sim_{MC}(L_i, L_{i+1}) \tag{3.27}$$

Because multiple lines of ancestry may exist between two concepts in a more complex ontology, we define $las(A,B)$ as the set of lines of ancestry between two concepts. For more than one line of ancestry, the one that yields the maximum MC will be decisive. Formally,

$$sim_{MC}^{las}(A,B) = \begin{cases} max_{L \in las(A,B)}(sim_{MC}(L)) & if\, las(A,B) \neq \emptyset \\ 0.0 & else \end{cases} \tag{3.28}$$

Finally, it should be noted that the MC-based derivation of probabilities for sim_{Resnik} and sim_{Lin} is based on the definition of subsumption relationships as described here. The only exception is the top node of an ontology, e.g., usually `owl:Thing`, which gets a probability of 1.

Common relationship: Two semantic concepts possess one or more common relatives, either ancestors or descendants. I.e., two concepts share the same type of ancestry with at least one related concept. LOG4SWS.KOM does not define an equivalent type of match. Given a common relative, the MC can be determined by computing the MCs between each of the two concepts and their respective relative and subsequently multiplying them. For more than one common relative, the maximal MC will be decisive. Under the assumption that a set of common ancestors $ancs(A,B)$ (excluding `owl:Thing`, which is a common ancestor of every OWL class) exists, we formally define

$$sim_{MC}^{ca}(A,B) = \begin{cases} max_{R \in ancs(A,B)}(sim_{MC}(R,A) * sim_{MC}(R,B)) & if\, ancs(A,B) \neq \emptyset \\ 0.0 & else \end{cases} \tag{3.29}$$

Likewise, under the assumption that a set of common descendants $descs(A,B)$ (excluding `owl:Nothing`, which is a common descendant of every OWL class) exists, we define

$$sim_{MC}^{cd}(A,B) = \begin{cases} max_{R \in descs(A,B)}(sim_{MC}(A,R) * sim_{MC}(B,R)) & if\, descs(A,B) \neq \emptyset \\ 0.0 & else \end{cases} \tag{3.30}$$

Because more than one of above relations might hold true in more complex ontologies, the overall MC is computed, using a maximization function:

$$sim_{MC}(A,B) = max(sim_{MC}^{eq}, sim_{MC}^{dsc}, sim_{MC}^{la}, sim_{MC}^{ca}, sim_{MC}^{cd}) \tag{3.31}$$

3.3.2 Deriving Level Weightings Using Regression Analysis

One of the central questions regarding service discovery is to which degree different abstraction levels of a service description need to be regarded in the matchmaking process. In LOG4SWS.KOM, it is possible to give weights to the different service abstraction levels. As will be seen in the evaluation, we tested different weightings. Nevertheless, the manual determination of such weightings is to some degree arbitrary. Furthermore, a particular weighting might be suitable for one service domain and completely wrong for another. To account for these drawbacks, COV4SWS.KOM applies an OLS estimator for the determination of optimal level weights.

We assume that the weighted linear combination of similarities on individual levels predicts the aggregated similarity of two operations, and thus, ultimately, two services. The process is based on the notion that a dependent variable $y^{a/b}$ – corresponding to the similarity of two operations a and b according to some numerical scale – can be derived through the linear combination of a set of independent variables $x_L^{a/b}$, corresponding to the individual similarity on a certain matching level L when matching a and b. The advantages and drawbacks of OLS have been discussed in Section 3.2.3. Again, as training data, a set of services is required along with a predefined similarity (or graded relevance) rating.

In the training phase, COV4SWS.KOM matches all pairs of operations in all service requests and offers. For each pair, it stores the computed similarity on each matching level and retrieves the predefined similarity of the two operations. If such figure is unavailable at the operation level, the corresponding value on the higher levels of interfaces or services will be used. The similarities constitute the vector of predictors (y) for the OLS process, with each entry corresponding to a pair of operations. The design matrix (X) is derived in the following manner: Each pair of operations yields one row with four entries, where each entry ($x_L^{a/b}$, i.e., in the example of two operations a and b) corresponds to the similarity on a certain level L.

An exemplary design matrix and vector of predictors are depicted in the following, with the first row containing the data for a pair of operations a and b (which is mutually relevant according to the binary scale shown in Equation 3.33), and the second row containing data for a pair a and c (which is not mutually relevant according to the binary scale):

$$X = \begin{pmatrix} x_{iface}^{a/b} & x_{op}^{a/b} & x_{in}^{a/b} & x_{out}^{a/b} \\ x_{iface}^{a/c} & x_{op}^{a/c} & x_{in}^{a/c} & x_{out}^{a/c} \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix} = \begin{pmatrix} 0.93 & 0.72 & 0.61 & 0.85 \\ 0.26 & 0.14 & 0.17 & 0.23 \\ \vdots & \vdots & \vdots & \vdots \end{pmatrix} \tag{3.32}$$

$$y = \begin{pmatrix} y^{a/b} \\ y^{a/c} \\ \vdots \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ \vdots \end{pmatrix} \tag{3.33}$$

Given a design matrix and vector of predictors, the standard OLS estimator can be applied in the following manner:

$$\hat{\beta} = (X'X)^{-1}X'y \tag{3.34}$$

$\hat{\beta}$ corresponds to the optimal estimate of level weights, whereas X denotes the design matrix and y the corresponding vector of predictors.

In order to derive actual level weights, we further process the vector. First, each negative entry is set to 0. This ensures that increasing similarities on certain levels do not have a negative impact on the aggregated similarity. The underlying idea is that it would be contradictive to common sense if higher similarity on one level resulted in diminished overall similarity. Second, the entries are adapted such that their sum matches the maximum relevance. This ensures that a pair of operations with perfect similarity on all matching levels is precisely assigned the actual maximum relevance. Notably, the handling of negative numbers is differing in LOG4SWS.KOM and COV4SWS.KOM – this can be traced back to the fact that negative numbers are actually allowed in LOG4SWS.KOM, but could lead to similarity numbers outside the range [0..1]. In contrast, in COV4SWS.KOM negative numbers are not valid as explained

above; in order to remove negatively weighted service abstraction levels, their corresponding weight is simply set to 0.

As an example, assume that the OLS estimator outputs the following estimate:

$$\hat{\beta} = \begin{pmatrix} 0.6 & 0.2 & 0.4 & -0.1 \end{pmatrix} \tag{3.35}$$

The last entry, -0.1, is consecutively set to 0, resulting in

$$\hat{\beta}^{+} = \begin{pmatrix} 0.6 & 0.2 & 0.4 & 0 \end{pmatrix} \tag{3.36}$$

Dividing all entries by the vector's sum, 1.2, and multiplying it by the maximum relevance – which we will assume to be 1 here, in accordance with a binary scale – results in the final vector of level weights w, namely

$$w = \begin{pmatrix} 0.5 & 0.1\overline{6} & 0.\overline{3} & 0 \end{pmatrix} \tag{3.37}$$

Based on the aforementioned order of entries, the weights for the individual levels of abstraction then correspond to $w_{iface} = 0.5$, $w_{op} = 0.1\overline{6}$, $w_{in} = 0.\overline{3}$ and $w_{out} = 0$. I.e., in this example, the interface level would be the strongest determinant of overall similarity, followed by inputs and the operation level. In line, outputs would be of no discriminatory value at all.

For evaluation purposes, cross-validation is applied, i.e., the design matrices and vectors of predictors are adapted as previously described in Section 3.2.3.

3.3.3 Implementation

Listing 3.6: Function calculateSimilarity (COV4SWS.KOM)

```
Input:      regComponent, offComponent: Service components of the same type                          1
Output:     calculateCovSimilarity, calculateNameSimilarity: Similarity between the both components  2
Variables:  reqComponent.mref, offComponent.mref: References to semantic concepts                    3
                                                                                                     4
function calculateSimilarity(reqComponent, offComponent) {                                           5
   if reqComponent.mref != null and offComponent.mref != null                                        6
      if ontologiesAvailable(regComponent.mref, offComponent.mref)                                   7
         return calculateCovSimilarity(reqComponent.mref, offComponent.mref)                         8
      else                                                                                           9
         return calculateNameSimilarity(regComponent.mref, offComponent.mref)                        10
      end if                                                                                         11
   else                                                                                              12
      return calculateNameSimilarity(regComponent.name, offComponent.name)                           13
   end if                                                                                            14
}                                                                                                    15
```

The implementation of COV4SWS.KOM shares certain parts with the implementation of LOG4SWS.KOM. However, the `calculateSimilarity` function has been adapted to employ the similarity measures introduced in this section instead of subsumption similarity. The resulting function is depicted in Listing 3.6. As it can be seen, the only difference affects Line 8. The algorithm for the determination of a similarity value is determined in Listing 3.7. Here, the similarity or semantic relatedness for two semantic concepts A and B is calculated. Function `calculateCovSimilarity` (Listing 3.7) determines which similarity function is invoked (Lines 6–22). While for sim_{Resnik} and sim_{Lin}, no caches have been implemented, for sim_{PL} and sim_{MC}, it is first checked if a similarity value is already available (Lines 13 and 18). For these measures, a newly computed similarity value is stored in the cache (Line 23). Notably, the determination of probabilities as necessary for sim_{Resnik} and sim_{Lin} is *not* part of these methods, but predetermined by the ontology store.

3.4 Evaluation and Discussion

To allow for a comparison with competing matchmaking approaches, LOG4SWS.KOM and COV4SWS.KOM have been extensively evaluated. The environment for this evaluation process is described in detail in

Listing 3.7: Function calculateCovSimilarity

```
1  Input:      A, B: Semantic concepts
2              measure: Similarity measure
3  Output:     sim: Similarity value of the concepts A and B
4
5  function calculateCovSimilarity(A, B, measure)
6     if measure = resnik
7        sim = calculateResnik(A, B)
8        return sim
9     else if measure = lin
10       sim = calculateLin(A, B)
11       return sim
12    else if measure = pathLength
13       if cache.contains(A, B, pathLength)
14          sim = cache.get(A, B, pathLength)
15       else sim = calculatePL(A, B)
16       end if
17    else if measure = MC
18       if cache.contains(A, B, MC)
19          sim = cache.get(A, B, MC)
20       else sim = calculateMC(A, B)
21       end if
22    end if
23    cache.store(A, B, measure, sim)
24    return sim
25 end function
```

Appendix B. The primary evaluation measures are taken from the field of IR and are based on *recall* and *precision* – more precisely, recall-precision curves, Average Precision (AP), R-Precision (RP), and Precision at k (P(k)) are applied. The latter three measures are considered in their macro-averaged form. An in-depth explanation of these measures is given in Appendix B.3. Furthermore, the runtime performance in terms of the Average Query Response Time (AQRT) is observed. The evaluation process as introduced in Appendix B makes it necessary to provide a service offer result set sorted in descending order based on the calculated similarity values. If some service offers share the same similarity value, we randomize the order of these specific service offers in the result set.

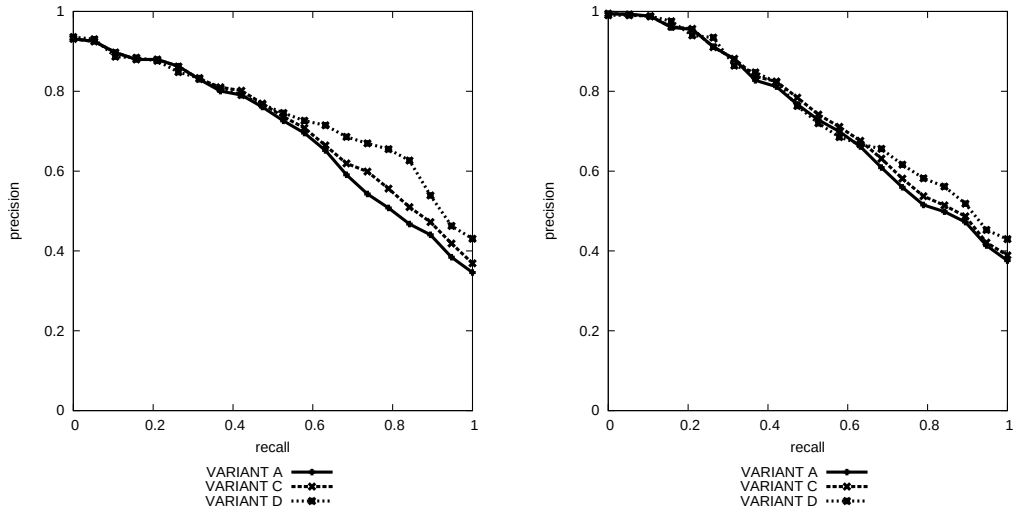
In the following, the evaluation results for LOG4SWS.KOM (Section 3.4.1) and COV4SWS.KOM (Section 3.4.2) will be presented, discussed, and compared (Section 3.4.3). The values presented in this section are all macro-averaged across all queries. Further evaluation results like, i.e., AP and RP *per query* are presented in Appendix C.

3.4.1 Evaluation of LOG4SWS.KOM

We have evaluated different versions of LOG4SWS.KOM in order to achieve comparability under varying circumstances and assess the impact of the different techniques applied in the matchmaker.

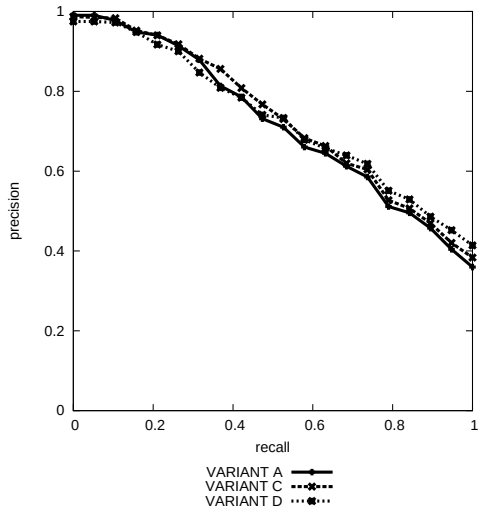
Per se, LOG4SWS.KOM is purely ontology-based, i.e., only the results from subsumption reasoning as well as the path length between concepts are used. However, SAWSDL-TC (cp. Section B.2.2), which is the applied test data set, only uses semantic annotations on parameter level – on interface and operation levels, no semantic description is given. So, in order to assess the ontology-based matching, it is necessary to restrict the matching to the service signature, i.e., input and output parameters. This is the first *version* tested; we assume that inputs and outputs are taken into consideration to the same degree, i.e., the weighting of inputs and outputs is 50% in each case. Important to note, this version does not apply any fallback strategy, even if some parameters lack semantic annotation – what is the case in SAWSDL-TC. These parameters will thus be assigned a similarity of 0 with all other parameters.

For the second and third version, interface and operation level are also taken into consideration. On these levels, the fallback strategy presented in Section 3.2.2 is applied in order to measure the native similarity between operations and interfaces, but also to parameters lacking semantic annotation. One question that arises here is how to weight the different service abstraction levels: In contrast to COV4SWS.KOM, LOG4SWS.KOM is not able to directly adapt to different qualities and impacts of service component descriptions. In order to meet the fact that the available semantic annotations of inputs and outputs should have more impact on the overall similarity than the weights for interfaces and operations,

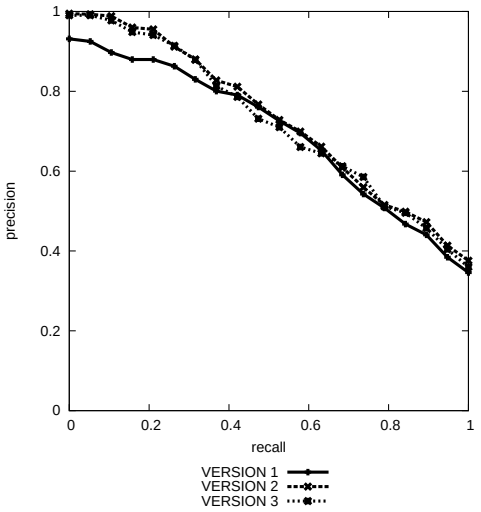


(a) Version 1

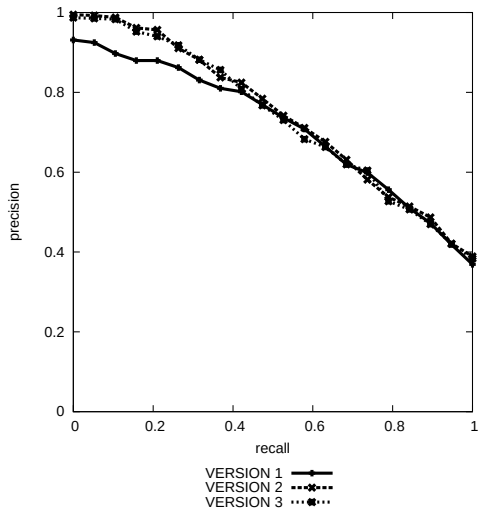
(b) Version 2



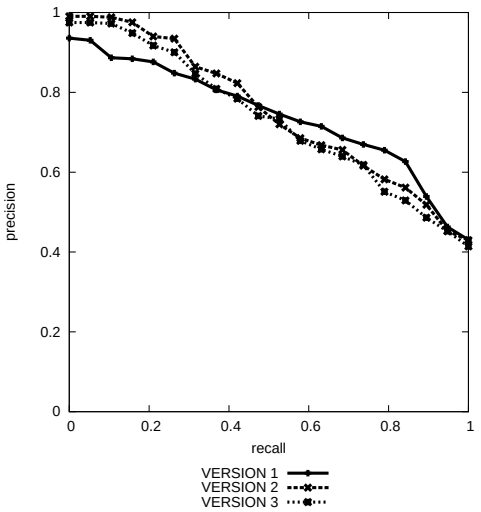
(c) Version 3



(d) Variant A



(e) Variant C



(f) Variant D

Figure 3.7: Performance of LOG4SWS.KOM (Recall-Precision Curves)

Table 3.4: Evaluation Results for Different Versions/Variants of LOG4SWS.KOM

#	Numerical Equivalents for Non-Inputs*	Weights for Interface/Operation/ Parameter Levels	AP	RP	P(5)	P(10)	AQRT (in ms)
1a	1/0.5/0.5/0	0.00/0.00/0.50	0.678	0.615	0.823	0.765	2137.37
1b	1/0.8/0.6/0	0.00/0.00/0.50	0.686	0.618	0.823	0.762	2187.69
1c	1/0.6/0.8/0	0.00/0.00/0.50	0.694	0.617	0.823	0.769	2235.48
1d	OLS applied	0.00/0.00/0.50	0.724	0.660	0.846	0.765	2217.52
2a	1/0.5/0.5/0	0.10/0.10/0.40	0.720	0.646	0.954	0.815	2316.90
2b	1/0.8/0.6/0	0.10/0.10/0.40	0.720	0.648	0.946	0.812	2287.62
2c	1/0.6/0.8/0	0.10/0.10/0.40	0.731	0.651	0.946	0.812	2405.13
2d	OLS applied	0.10/0.10/0.40	0.739	0.681	0.962	0.835	2376.08
3a	1/0.5/0.5/0	0.25/0.25/0.25	0.709	0.645	0.938	0.815	2253.06
3b	1/0.8/0.6/0	0.25/0.25/0.25	0.716	0.646	0.938	0.823	2332.83
3c	1/0.6/0.8/0	0.25/0.25/0.25	0.726	0.657	0.938	0.838	2280.19
3d	OLS applied	0.25/0.25/0.25	0.722	0.672	0.931	0.842	2462.88

* For exact/super/sub/fail. For inputs, the weighting of super&sub is reversed for Variants A-C.

the latter are weighted with 10% while the weightings for inputs and outputs are 40% each in the second version. For the third version, we decided to weight interfaces, operations, inputs, and outputs the same, i.e., 25% each.

For every *version*, we evaluate four different *variants* – the first variant makes use of the manually assigned numerical representations of distinct DoM levels as presented by Syeda-Mahmood et al., i.e., the numerical value for an *exact* match is a 1.0, for both *super* and *sub* 0.5, and for a *fail* 0.0 [244]. For the second and third variant, we have *manually* derived different numerical equivalents for the discrete DoMs presented in Equation 3.7. A description how these values have been determined is presented in Appendix B.2.3. Variant B follows the ranking by Paolucci et al., i.e., *exact* > *plugin* > *subsumes* > *fail* (cp. Section 3.2.1) [201]. This is done by setting the equivalents to 0.8 for *super* and 0.6 for *sub* for outputs. For inputs, these values have been reversed, corresponding to 0.6 and 0.8 for *super* and *sub* matches respectively. An *exact* match is a 1.0, and a *fail* results in a numerical equivalent of 0.0. As it was mentioned before, Cardoso respectively Bellur et al. reverse the ranking of more generic and more specific semantic concepts – their approaches favor service offers that expect more generic input over those with a more specific input, while the opposite is true for outputs [17, 45]. This is reproduced in the third variant – hence, a *super* gets a numerical equivalent of 0.6, a *sub* is equivalent to 0.8. Again, for inputs, the ranking and therefore the numerical equivalents are reversed. Finally, the fourth variant applies OLS-based numerical equivalents to DoM levels as presented in Section 3.2.3.

Figure 3.7 shows the resulting recall-precision curves for the three versions respectively Variants A, C, and D of LOG4SWS.KOM applied; the AP, RP, P(5), and P(10) values, and AQRT can be found in Table 3.4. For the purposes of Figure 3.7, Variant B has been omitted but as can be seen from the numbers in Table 3.4, the basic assumption regarding DoM ranking of Cardoso and Bellur et al. leads to better evaluation results for almost every single number. Hence, we can deduct that the ranking of these authors is more realistic than that of Paolucci et al.

As it can be seen from the figures, the differences in evaluation results are relatively high for the version that only applies to the service signature and still recognizable for the other versions. A Friedman test has been conducted to detect statistical significant differences between the results of different evaluation runs (cp. Appendix B.2). For all three versions, the results between all four variants differ highly significant, while this is not the case for every pair of variants from a certain version. A more detailed presentation of the Friedman test results is given in Appendix C.3.

The results from the Friedman tests indicate that for Version 1, the differences in results are due to the integration of OLS-based numerical equivalents, while for Version 2 this applies only to some extent as there is no statistically significant difference between Variants 2c and 2d. For Version 3, there is no statistically significant difference between Variant 3d and any other variant. However, it should be kept in mind that regarding Version 2 and 3, the variants without applied OLS already provide very good results. As we will see from the comparison in Section 3.4.3, until now, URBE [212] has so far provided the best matchmaking results with an AP of 0.727. This value is almost met by all variants of Versions 2 and 3 and even exceeded by Variants 2c (AP=0.731) and 2d (AP=0.739).

Version 1, which is purely ontology-based and limited to service signature matching, delivers its best AP for the variant with OLS applied (Variant 1d). As it can be seen in the recall-precision curves in Figure 3.7a and the numbers in Table C.10, all three depicted variants of Version 1 start with a comparably mediocre macro-averaged precision of about 0.89 for a recall level of 0.105, which declines in different slopes to a recall level of 1.0. Variant 1d performs best of all 12 variants tested with a precision of 0.431 for a recall level of 1.0. As we will see in Section 3.4, this is an extraordinary good result.

As can be seen from Figures 3.7b and 3.7c, the syntactic information from the interfaces' and operations' names actually leads to an improvement in matchmaking results. Variant 2a shows an AP of 0.720 while Variant 2d (with OLS) leads to an AP of 0.739. For Variant 2d, the macro-averaged precision is at 0.935 (Variant 2a: 0.912) for a recall level of 0.263 and declines to 0.430 (Variant 2a: 0.376) for recall = 1.0. The values for Variant 2c are positioned between 2a and 2d with higher AP values for medium recall levels.

The results for the altered weights for interface and operation levels show that the assumption that these levels should be weighted less than the semantically annotated inputs and outputs has proven right. For a recall level of 0.263, the average precision is about 0.914 (for Variant 3a) respectively 0.900 (for Variant 3d). For recall level 1.0, the average precision is 0.360 (3a) respectively 0.414 (3d). Again, the values for Variant 3c can be found between 3a and 3d with higher AP values for medium recall levels. Nevertheless, Variant 3c (AP=0.726) slightly outperforms Variant 3d (AP=0.722) regarding the overall AP.

As shown in Figures 3.7a-3.7c, especially for Versions 1 and 2, the improvement of matchmaking results based on the application of OLS is primarily attributed to the better performance for higher recall levels (>0.5 for Version 1 and >0.7 for Version 2). Up to these levels, results are comparable. For Version 3, the three depicted variants perform quite similar with generally slightly lower results for Variant 3a and slightly better values for the middle recall levels for Variant 3c, which ultimately leads to the point that Version 3 is the only version where Variant D does not offer the best overall AP.

Regarding RP, P(5), and P(10) precision values, to the best of our knowledge, there is no matchmaker we can use for comparison. However, URBE also presents P(k) values for a test data set related to SAWSDL-TC [212]. Here, LOG4SWS.KOM clearly outperforms URBE's P(5) value of 0.867 with values bigger than 0.930 in Version 2 and 3. Regarding P(10), these versions still provide better numbers than URBE, however, the gap is smaller. Regarding RP, URBE provides a value of 0.651 which is exceeded by Variants 1d (0.660), 2d (0.681), and 3d (0.672).

If comparing the three versions of LOG4SWS.KOM with each other regarding RP, P(5), and P(10), Versions 2 and 3 provide similar evaluation results while outperforming Version 1. Regarding P(5), Version 2 provides the best results followed by slightly lower results in Version 3 and clearly lower results for Version 1 which also performs worst regarding P(10). Here, Version 3 offers the best results while Version 2 provides slightly worse numbers. Again, this highlights that Version 1 performs worse for lower recall levels than the other versions.

We infer the following conclusions from the results: First of all, the application of OLS improves matchmaking results noticeably (Versions 1 and 2) respectively does not lead to a significant decrease in results (Version 3). Second, the integration of similarity values from interface and operation levels might be suitable to increase the precision values. As it can be seen from all variants (Figures 3.7d-3.7f), the integration of the interface and operation levels lead to a significant improvement of precision values for low recall levels (recall $\leq$ 0.30). Thus, we can assume that the decision whether a service is relevant to a particular query in SAWSDL-TC cannot be traced back only to the message parameters. The integration of other service abstraction levels than the service signature seems to be a sufficient way to improve matchmaking results on these levels. However, this integration helps only to some degree if the description information on these levels is not sufficient. In the evaluation at hand, semantic annotations

were not available on interface and operation levels; as a result, a relatively low weighting of these levels improves matchmaking results a lot (Version 2), but if these levels are weighted too much, evaluation numbers start to decrease (Version 3).

Apart from the evaluation regarding IR performance measures, the runtime performance in terms of AQRT has also been evaluated. The AQRT significantly depends on optional preprocessing steps which are presented in Section B.5. Important to note, the derivation of weightings based on OLS estimation is conducted at runtime, because cross-validation requires knowledge of the respective current query. An overview of the macro-averaged AQRT (median of the test runs conducted) for the different versions and variants of LOG4SWS.KOM can be found in Table 3.4. As it can be seen, Variants A-D feature relatively similar *median* AQRTs. Hence, it can be assumed that the identification of numerical equivalents based on OLS, which is conducted during runtime, does not extend the runtime performance to a large degree. However, Table C.8 shows that the *mean* AQRT of Variant D is indeed a little bit higher than those of Variants A–C. Regarding the different versions, Version 1 performs slightly better as it does not require computations on interface and operation level, Versions 2 and 3 perform a little bit worse. However, the differences are too small to be of practical relevance. As the processing of SAWSDL service descriptions to ATSM models is the most time-consuming part of the query processing (cp. Section B.5), taking about 80% of the overall time, the runtime performance could be further improved if this processing was accelerated. A more detailed presentation of the AQRTs can be found in Appendix C.4. A comparison of these results with the results from COV4SWS.KOM and other matchmakers can be found in Section 3.4.3.

3.4.2 Evaluation of COV4SWS.KOM

Analog to the different versions and variants evaluated for LOG4SWS.KOM, different versions and variants of COV4SWS.KOM have been tested. As in Version 1 of LOG4SWS.KOM, Version 1 of COV4SWS.KOM also incorporates only matching on parameter level, i.e., inputs and outputs are weighted 50% each. Similarity measurement is only applied to semantically annotated parameters – hence, it is not necessary to make use of the fallback strategy. The second and third version reprise the weights introduced in Versions 2 and 3 of LOG4SWS.KOM: In Version 2 interfaces and operations are weighted to a lesser extent (i.e., 10% each) and input and output parameters are weighted to a larger extent (i.e., 40% each). Version 3 applies an equal weighting of interfaces, operations, inputs, and outputs (i.e., 25% each). As COV4SWS.KOM does not need to derive numerical equivalents for distinct DoMs, it is possible to apply OLS-based weighting from another point-of-view – in this case, this is the weighting of service abstraction levels. Hence, Version 4 is applied which is based on automatic weighting of these levels.

For each *version*, we evaluate six *variants*. While the variants in LOG4SWS.KOM concern different numerical equivalents for subsumption-based DoMs, variants in COV4SWS.KOM are based on the different similarity measures and probability values presented in Section 3.3:

Variant A is based on sim_{Resnik}, i.e., the similarity metric proposed by Resnik, and makes use of probability values derived from the SAWSDL-TC corpus.

Variant B is also based on sim_{Resnik}, but makes use of the probability values derived with MC.

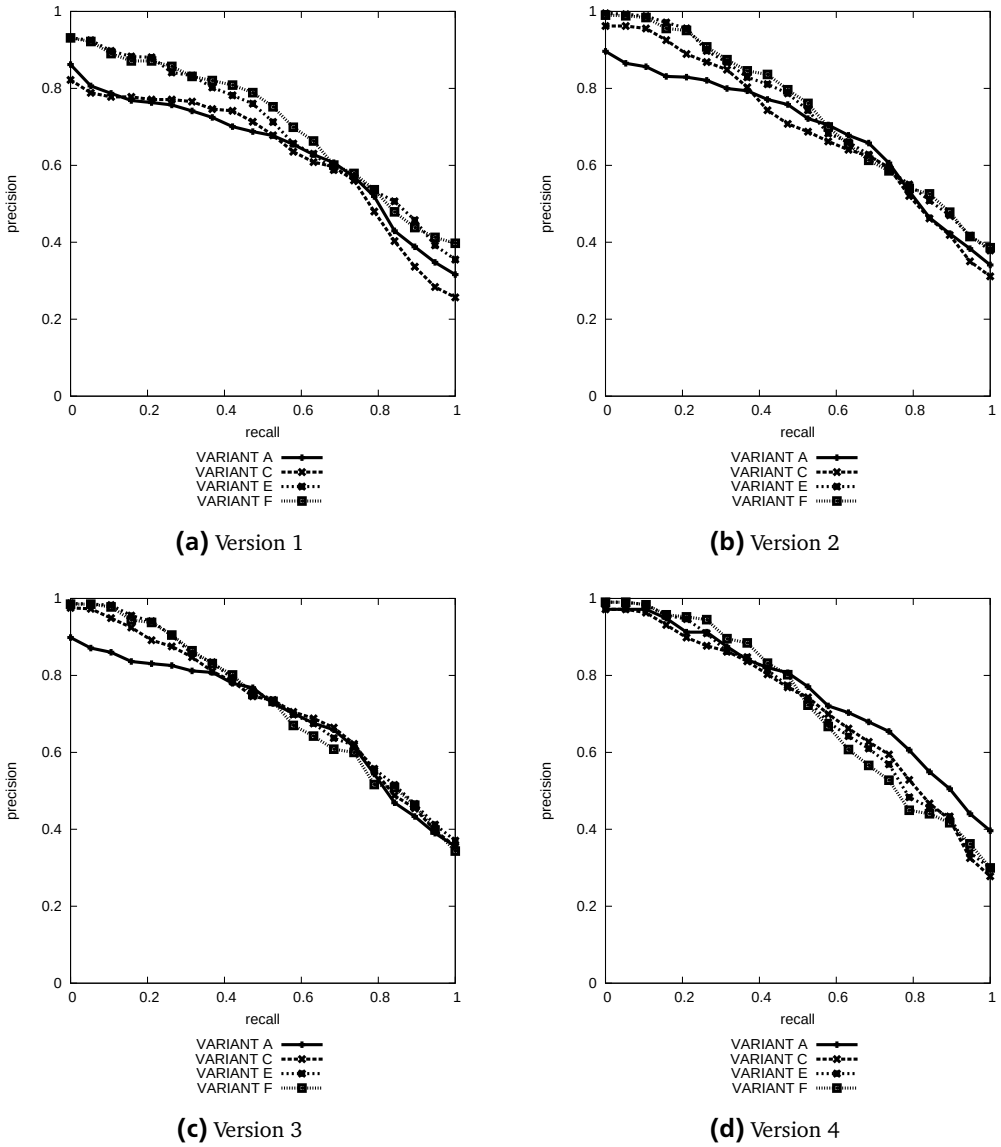
Variant C is based on sim_{Lin}, i.e., the similarity metric proposed by Lin, and makes use of probability values derived from the SAWSDL-TC corpus.

Variant D is also based on sim_{Lin}, but makes use of the probability values derived with MC.

Variant E is a reference value making use of sim_{PL} as proposed by Rada et al., i.e., the inverted path length is used as similarity metric.

Variant F is based on sim_{MC} as introduced in Section 3.3.1.

Figures 3.8 and 3.9 show the recall-precision curves for the four versions of COV4SWS.KOM; Table 3.5 shows the AP, RP, P(5), and P(10) values as well as the AQRT. For the purposes of Figure 3.8, Variants B and D have been omitted. However, as can be seen from Table 3.5, the MC-based variants of sim_{Resnik} and sim_{Lin} mostly perform slightly worse than the corpus-based variants. Exceptions will be discussed in the following.



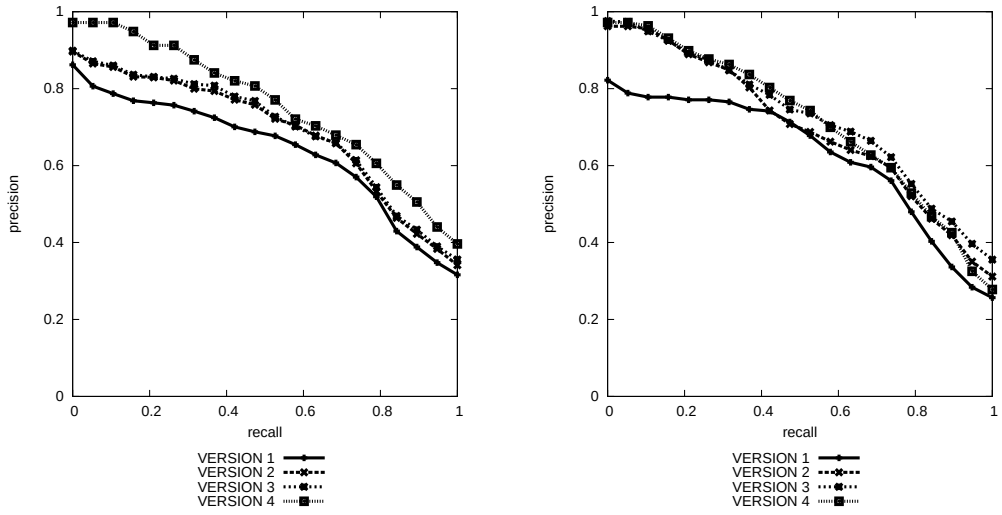
(a) Version 1 **(b)** Version 2

(c) Version 3 **(d)** Version 4

Figure 3.8: Performance of COV4SWS.KOM (Recall-Precision Curves) – Versions

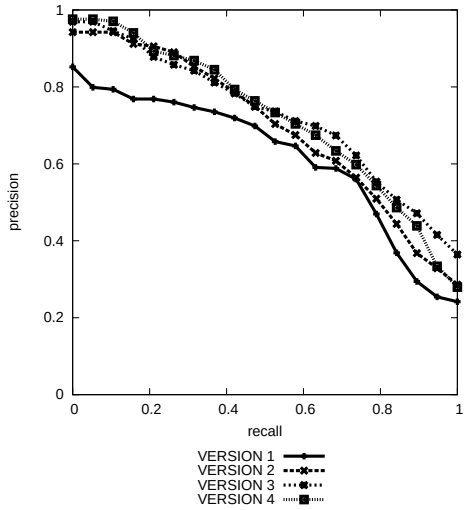
As it can be seen from Figure 3.8, the differences in evaluation results are obvious, especially for Versions 1, 2, and 4. On the contrary, Version 3 shows relatively similar recall-precision curves. Again, Friedman tests have been conducted to detect statistical differences between the results of different evaluation runs (cp. Appendix C.3). These tests confirm the curves in terms of heterogeneity: While for Version 3, there is no statistical difference between evaluation results of all six variants, there is such a difference for all other versions. As it has been the case for LOG4SWS.KOM, not every pair of variants from a certain version features statistically significant differences. This makes it generally difficult to draw definite conclusions from the data. The results from the Friedman tests indicate that for Versions 1 and 2, differences for sim_{Resnik} and sim_{Lin} are rather small, while for Version 3, there is a significant difference between the MC-based Variants B and D. For Version 4, the results differ statistically significant as it is already indicated by the large gap between evaluation results between Variants 4a/4c and Variants 4b/4d.

The pure ontology-based Version 1 possesses the worst evaluation results, as for every single performance metric, the other three versions feature better results. There is only one negligible exception as RP of Variant 1f (0.634) is 0.001 higher than its equivalent for Variant 4f. As can be seen from Figure 3.8a as well as from the relatively low P(5) and P(10) values, this can be primarily traced back to the low precision results for low recall levels, while for Variants D, E, and F, the precision values are acceptable

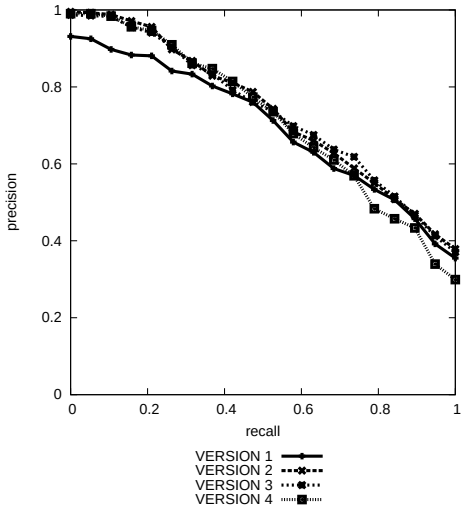


(a) Variant A

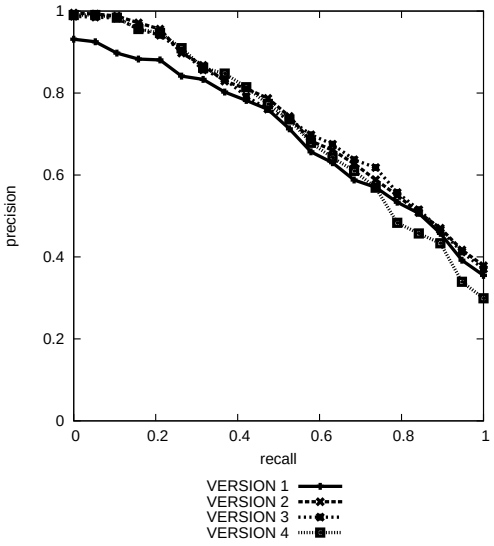
(b) Variant B



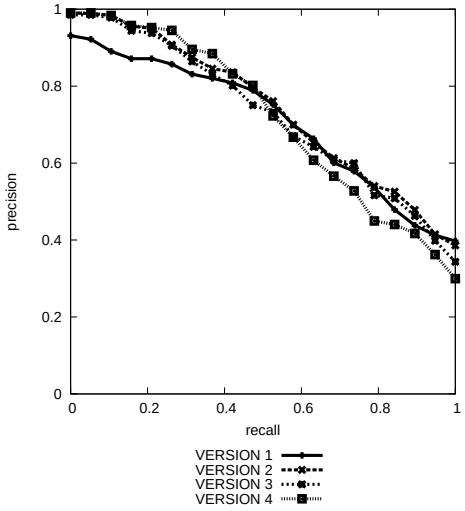
(c) Variant C



(d) Variant D



(e) Variant E



(f) Variant F

Figure 3.9: Performance of COV4SWS.KOM (Recall-Precision Curves) – Variants

Table 3.5: Evaluation Results for Different Versions/Variants of COV4SWS.KOM

#	Similarity Metric Applied	Weights for Interface/Operation/ Parameter Levels	AP	RP	P(5)	P(10)	AQRT (in ms)
1a	$sim_{Resnik}(Corp)$	0.0/0.0/0.5	0.616	0.552	0.715	0.673	2322.19
1b	$sim_{Resnik}(MC)$	0.0/0.0/0.5	0.586	0.558	0.715	0.650	2303.15
1c	$sim_{Lin}(Corp)$	0.0/0.0/0.5	0.605	0.571	0.692	0.685	2366.33
1d	$sim_{Lin}(MC)$	0.0/0.0/0.5	0.596	0.558	0.723	0.662	2389.38
1e	sim_{PL}	0.0/0.0/0.5	0.682	0.608	0.823	0.754	2211.67
1f	sim_{MC}	0.0/0.0/0.5	0.687	0.634	0.815	0.765	2099.77
2a	$sim_{Resnik}(Corp)$	0.1/0.1/0.4	0.672	0.599	0.785	0.719	2496.46
2b	$sim_{Resnik}(MC)$	0.1/0.1/0.4	0.642	0.593	0.769	0.700	2642.29
2c	$sim_{Lin}(Corp)$	0.1/0.1/0.4	0.687	0.626	0.885	0.785	2562.00
2d	$sim_{Lin}(MC)$	0.1/0.1/0.4	0.683	0.618	0.885	0.796	2729.56
2e	sim_{PL}	0.1/0.1/0.4	0.728	0.655	0.946	0.804	2445.50
2f	sim_{MC}	0.1/0.1/0.4	0.730	0.667	0.931	0.823	2376.75
3a	$sim_{Resnik}(Corp)$	0.25/0.25/0.25	0.679	0.611	0.785	0.719	2592.50
3b	$sim_{Resnik}(MC)$	0.25/0.25/0.25	0.648	0.589	0.777	0.696	2528.52
3c	$sim_{Lin}(Corp)$	0.25/0.25/0.25	0.711	0.673	0.900	0.823	2714.12
3d	$sim_{Lin}(MC)$	0.25/0.25/0.25	0.714	0.671	0.885	0.812	2696.08
3e	sim_{PL}	0.25/0.25/0.25	0.726	0.658	0.915	0.842	2325.12
3f	sim_{MC}	0.25/0.25/0.25	0.713	0.657	0.915	0.823	2352.90
4a	$sim_{Resnik}(Corp)$	OLS applied	0.746	0.686	0.923	0.835	2726.54
4b	$sim_{Resnik}(MC)$	OLS applied	0.733	0.678	0.931	0.815	2662.79
4c	$sim_{Lin}(Corp)$	OLS applied	0.704	0.655	0.915	0.819	2810.60
4d	$sim_{Lin}(MC)$	OLS applied	0.707	0.651	0.931	0.819	2786.58
4e	sim_{PL}	OLS applied	0.706	0.628	0.915	0.796	2185.35
4f	sim_{MC}	OLS applied	0.702	0.633	0.915	0.831	2446.00

for higher recall levels (cp. Figure 3.9). For Variants A to C, Version 1 features the worst precision values for nearly all recall levels. The precision values for low recall levels for Version 1 approve the finding for LOG4SWS.KOM that for these recall levels, a pure semantic description of service components is not sufficient – even though COV4SWS.KOM applies different similarity measures for semantics.

As it has been already observed in the evaluation of LOG4SWS.KOM, the consideration of further service abstraction levels in Versions 2 and 3 leads to an improvement of evaluation results. However, in contrast to LOG4SWS.KOM, the higher weights for interface and operation levels in Version 3 (in comparison to Version 2) do not generally lead to a decline in evaluation results: Regarding the AP, RP, P(5), and P(10), Variants 3a-d generally outperform Variants 2a-d, with RP and P(10) for Variants 2b/3b being the only (insignificant) exception. For Variant E, the values vary without very large differences, for Variant F, Version 2 clearly provides better results than Version 3.

Regarding OLS-based weighting of service abstraction levels in Version 4, the evaluation has led to mixed results. While for Variant 4a very good results have been achieved, the integration of automatic level weighting does lead to a degradation in evaluation results for Variants E and F. For Variants C and D, OLS leads to mediocre results, which are not as good as results for Variants 3c and 3d.

For Version 1, sim_{Resnik} and sim_{Lin} feature very similar evaluation results, if comparing the corpus-based and MC-based values pairwise. There is no clear pattern which variants perform better, while for Versions 2 and 3, sim_{Lin} outperforms sim_{Resnik} for all numbers observed. While there is no observable

improvement of evaluation results from Version 2 to 3 for sim_{Resnik}, the values of sim_{Lin} are clearly better for Version 3 if compared to Version 2 and also outperform results of OLS-based level weighting in Version 4 (cp. Table 3.5). The amendable performance of sim_{Resnik} can be traced back to the fact that these versions incorporate different service abstraction levels; as sim_{Resnik} is not normalized to the range [0..1], this metric is given a disproportionately high weight, leading to overall worse evaluation results. However, this issue is solved by the OLS-based weighting of abstraction levels in Version 4; here, the non-normalization is confronted by the automatic weighting. Hence, sim_{Resnik} is adjusted to the similarity measures from other service abstraction levels. This leads to the overall best AP and RP values for Variant 4a – even if compared to LOG4SWS.KOM. As can be seen from Figure 3.8d, these values are the result of an extraordinary good performance for low and middle recall levels.

The variety and number of evaluation results makes it difficult to derive generally valid conclusions. However, there are certain notable outcomes: First of all, the heterogeneity of results shows that there is no generally best matchmaking method for Web services – if reducing the evaluation results to the similarity measures applied, path length, mutual coverage, and the similarity measure of Resnik all lead to the best values under certain restrictions. As a second conclusion, we deduct that OLS-based optimization of service abstraction level weights does not necessarily lead to an improvement of matchmaking results. In fact, only Variants A and B generally benefit in Version 4. Interestingly, OLS-based weights lead to an improvement of P(5) for every case, i.e., for low recall levels. Third, purely service signature matching as applied in Version 1 leads to the worst evaluation results, confirming the results from LOG4SWS.KOM. Finally, we would like to dwell on sim_{MC} which is – in contrast to the other applied similarity metrics – not a general similarity metric, but depending on certain assumptions typical to service matchmaking. Variant F respectively sim_{MC} leads to generally good results and features the best AP and RP values for Version 2. Hence, we deduct that this similarity metric is worth pursuing.

All things considered, COV4SWS.KOM performs very well if the fallback strategy is applied. In our opinion, the results show that the usage of OLS and metrics, which are usually used in order to determine semantic relatedness, might be a strategy in order to improve ontology-based matchmaking results and overcome the issue that current ontologies (like those from SAWSDL-TC) are often only simple taxonomies that do not rely on advanced features provided by, e.g., OWL DL and that only coarse-grained concept descriptions are available [132].

Again, the runtime performance of the matchmaker at hand in terms of AQRT has been evaluated. An overview of the macro-averaged AQRT (median of the tests conducted) for the different versions and variants of COV4SWS.KOM can be found in Table 3.5. As it can be seen, Version 1 generally features the lowest AQRT values, Versions 2 and 3 feature similar values, and Version 4 features the highest AQRTs. Similar to the values observed for LOG4SWS.KOM, the differences can be attributed to the weightings of service abstraction levels. While Version 1 incorporates the service signature, Version 2 and 3 also compute similarity values for the interface and operation levels, causing higher matchmaking runtime. In Version 4, it is first necessary to determine the weights using OLS, consequently, the actual weighting is conducted, leading to the highest AQRT values. Regarding the different variants, Variants A and B perform quite similar, as do Variants C and D. This is not surprising, as these pairs of variants make use of exactly the same methods but operate on different probability values. The last-mentioned variants perform slightly worse, as the similarity computation is a little bit more complex (cp. Equations 3.18 and 3.20). Variants E and F possess the best AQRT values, as they make use of caches. Again, the differences are quite small. The same conclusions can be drawn as for LOG4SWS.KOM – as the largest part of the matchmaking process duration is attributed to the transformation of the query from SAWSDL to ATSM, the runtime could be primarily accelerated by improving this transformation. Further runtime evaluation results can be found in Appendix C.4.

3.4.3 Discussion

In this section, we will compare the evaluation results from LOG4SWS.KOM and COV4SWS.KOM with each other and with other matchmaking approaches for SAWSDL. The presented matchmakers only differ regarding the ontology-based deduction of similarity values and how the weighting of service abstraction levels is achieved. As an overview, Table 3.6 recapitulates the evaluation results for the best variants of both matchmakers. Additionally, the table shows evaluation results of further matchmakers for SAWSDL evaluated using SAWSDL-TC 1, i.e., the WSDL 1.1-based version of SAWSDL-TC (cp. Appendix B.2.2).

Table 3.6: Comparison of AP of Matchmakers for SAWSDL

	Adaptive	AP	RP	P(5)	P(10)
SAWSDL-M0 [132, 140]	NO	0.400	N/A	N/A	N/A
SAWSDL-MX2 [140, 142]	automatically	0.679	N/A	N/A	N/A
COV4SWS.KOM (Variant 1f)	manually	0.687	0.634	0.815	0.765
LOG4SWS.KOM (Variant 1d)	automatically	0.724	0.660	0.846	0.765
URBE [142, 212]	manually	0.727	0.651	0.867	0.796
COV4SWS.KOM (Variant 2f)	manually	0.730	0.667	0.931	0.823
LOG4SWS.KOM (Variant 2d)	automatically	0.739	0.681	0.962	0.835
COV4SWS.KOM (Variant 4a)	automatically	0.746	0.686	0.923	0.835

Regarding Versions 1–3, LOG4SWS.KOM generally outperforms COV4SWS.KOM. However, COV4SWS.KOM Variant 4a is overall the best matchmaker regarding AP and RP – at least regarding the applied test data set. While in COV4SWS.KOM, "only" Variants 4a and 4b benefit from the application of OLS, for LOG4SWS.KOM, OLS generally leads to improved matchmaking results, even though the AP of Variant 3c is slightly worse than those of Variant 3d.

On the one hand, the best values for COV4SWS.KOM Variants C–F have been achieved when the service abstraction level weights have been manually tuned and set. On the other hand, such a manual task could lead to an extensive amount of work, especially if the regarded service landscape changes quickly. Even though COV4SWS.KOM provides sometimes worse matchmaking results, it comes with the benefit of direct adaptation to differing degrees of semantic and syntactic description "richness" for certain service abstraction levels. Such an adaptation is also conducted in LOG4SWS.KOM – however, not directly.

All things considered, both matchmakers perform very well whether or not the fallback strategy or adaptation mechanisms are applied. To the best of our knowledge and compared with the results of the S3 Contest 2009 [142], LOG4SWS.KOM and COV4SWS.KOM provide the best matchmaking results of any SAWSDL matchmaker so far. In contrast to URBE and SAWSDL-MX2, our matchmakers make use of text similarities only as a substitute if an ontology-based matchmaking cannot be carried out. Especially the logic-based version of LOG4SWS.KOM features a higher certainty than the IR methods used in URBE and SAWSDL-MX2. Furthermore, the purely ontology-based matchmakers, namely LOG4SWS.KOM Variant 1d and COV4SWS.KOM Variant 1f, provide much better results than SAWSDL-M0, which is, to the best of our knowledge, the only other purely ontology-based SAWSDL matchmaker with comparable evaluation results [132, 140].

When comparing the recall-precision curves of LOG4SWS.KOM and COV4SWS.KOM with those of URBE and SAWSDL-MX2 [142], it is eye-catching that our matchmakers provide much better precision results for high recall levels. For example, SAWSDL-MX and URBE feature an AP of about 0.20 for recall = 1.0 while the best variant of LOG4SWS.KOM (2d) features a value of 0.430 and Variant 4a of COV4SWS.KOM still features a value of 0.404. Nevertheless, the RP, P(5), and P(10) values show that the matchmakers presented in this thesis are also able to improve matchmaking results for low and middle recall levels (compared to URBE).

In addition to the evaluation results, we want to highlight and discuss the impact and consequences of applying numerical subsumption DoMs respectively numerical values based on semantic relatedness instead of the usually applied subsumption matching similarity values as presented by Paolucci et al. and adapted by, e.g., [45, 140, 164, 238]. Usually, subsumption matching applies DoMs for discrete elements in a service description and defines the *minimum* DoM found as the overall service (or operation) DoM. This leads to a quite coarse-grained, discrete scale of possible service DoMs. As Fernández et al. state, these DoMs are not sufficiently fine-grained [82]. The discrete scale of possible service DoMs further implies that in order to further rank service offers based on a service request, additional techniques like, e.g., text similarity needs to be applied. A continuous, numerical measure like those applied in LOG4SWS.KOM and COV4SWS.KOM allows for a more precise ranking of services. In COV4SWS.KOM, the numerical similarity is directly computed which permits easy combination with other measures.

LOG4SWS.KOM and COV4SWS.KOM implicitly account for differing path lengths between concepts and thereby efficiently punish overly generic semantic annotations, which may otherwise lead to wrong search results [17]. Furthermore, we do not explicitly have to treat inputs and outputs differently. Related work often makes such a distinction, which also requires somewhat arbitrary assumptions regarding a ranking, depending on the type of subsumption relation (cp. Section 3.1.2). In COV4SWS.KOM, semantic relatedness is applied which supersedes the distinction between inputs and outputs and generally assigns the same similarity value to a pair of concepts, regardless of the matching level they stem from. In LOG4SWS.KOM, we also use a generic definition of subsumption matching types. The decision whether more generic or more specific concepts should be ranked higher on certain levels is automatically derived through the OLS estimator and may well differ between, e.g., inputs and outputs. Additionally, to account for user preferences, such rankings may be manually specified in LOG4SWS.KOM by assigning the appropriate weights.

We also would like to discuss a potential shortcoming of our matchmakers: If following the approach presented by Paolucci et al. [201], the computed DoM for any operation can be assumed as a guaranteed lower bound of similarity for the request [140]. With the average-based DoM computed by our matchmakers, such a lower bound is not guaranteed. However, pros and cons of having such a lower bound have to be weighted – on the one hand, this guarantees a certain degree of similarity for the request. But on the other hand, this approach is quite prone to outliers, i.e., one very low DoM has a very large impact on the overall DoM. Here, a non-discrete scale which makes it possible to derive an average DoM is certainly helpful. Hence, it is worth discussing if a certain degree of similarity needs to be guaranteed – notably, URBE, which is currently providing the best SAWSDL matchmaking results in terms of AP (apart from LOG4SWS.KOM and COV4SWS.KOM) does also not make use of such a lower bound [212]. Another approach, which makes use of implicit semantics and does not guarantee a global DoM is *iMatcher* by Kiefer and Bernstein [123]. Sheth et al. argue that implicit semantics should be generally regarded and a restriction to formal semantics and DL will limit the potential of the Semantic Web [227].

In our opinion, in a context where no minimal DoM is given in a service request – what is always the case if formulating queries using a "query by example" approach as it is currently done in the service matchmaking research community (cp. Section 4.1) – it is legitimate to abstain from guaranteeing such a lower bound of DoM. If a minimal DoM is defined in a query, it is necessary to consider this during matchmaking: In all approaches presented in this thesis, this can be done by adding certain constraints to the matchmaker. However, this is not required in common evaluation approaches for (SAWSDL-based) service matchmakers, where the service request is given as a query by example.

Furthermore, we would like to dwell on the runtime performance of our matchmakers. Even though runtime performance has only been paid little attention when designing the matchmakers, the evaluation results are promising. A comparison of the runtime performance with that of other matchmakers like URBE is difficult, as we were not able to conduct a runtime evaluation of the other matchmakers on the same machine; here, the 2010 edition of the S3 Contest will deliver resilient values. However, COM4SWS, which was a preliminary version of LOG4SWS.KOM and COV4SWS.KOM without, e.g., the caching mechanisms, had an AQRT of 6.14 seconds in the S3 Contest 2009. In comparison, SAWSDL-MX2 needed an AQRT of 7.9 seconds and URBE of 19.96 seconds [142]. Through some changes in the design – especially the introduction of caches and the representation of SAWSDL files as ATSM models – we were able to speed up the AQRT by a huge factor. Thus, COV4SWS.KOM and LOG4SWS.KOM also provide competitive evaluation results regarding the runtime performance. However, these results have to be traced back to the serialization of services into ATSM and the caching mechanisms, not to the matchmakers themselves. In an application scenario where services are registered or stored in a service catalogue, this is a valid assumption.

As a final consideration, it should be discussed for which domains COV4SWS.KOM or LOG4SWS.KOM might be better suitable. The biggest advantage of COV4SWS.KOM is the direct adaptation to different usefulness of service component descriptions on different service abstraction levels. Regarding LOG4SWS.KOM, such an adaptation is indirectly supposable as the OLS adaptation is independently conducted for each abstraction level. On the one hand, the weighting of service abstraction levels has still to be adapted manually; on the other hand, LOG4SWS.KOM will nevertheless be adapted to the regarded service domain. Furthermore, Variant 4a of COV4SWS.KOM features very good results for low recall levels, which are most likely more important to the user than a high AP.

As a conclusion, we recommend to make use of COV4SWS.KOM if it is possible to clearly distinguish between different service domains or which service ontology is used in (part of) a repository. As an example, in the energy domain, the Common Information Model (CIM) presents an extensive ontology available in OWL which could be used to semantically describe corresponding services [248]. In the e-commerce domain, *GoodRelations* by Hepp or *eClassOWL* could be used [102, 104]. In other domains, such an ontology is missing. Differences between services do not need to come from the consideration of a certain business domain, but might also arise from the usage of different service ontologies in different domains (with one popular example being WSMO-Lite [251]). If services from clearly distinguishable domains would be available in one (potentially distributed) service registry, it seems reasonable to make use of differently adapted/configured matchmakers based on COV4SWS.KOM. Still, it is a requirement that services in such a domain possess comparably useful component descriptions on all levels.

Regarding the evaluation results, it should be mentioned that the results will most certainly vary if applying a different test data set. However, SAWSDL-TC is to the best of our knowledge the only comprehensive and publicly available test data set for SAWSDL and therefore also used in the annual S3 Contest [142] and by most researchers in this field (e.g., [123, 140, 212]).

3.5 Related Work

As it can be seen from the matchmaking approaches that have already been referenced in this chapter, numerous approaches have been proposed in recent years. These approaches address different aspects of matchmaking, different service formalisms, or rely on different assumptions regarding the depth and richness of semantic service descriptions. As the related work is very extensive, the following examination will be limited to approaches which have heavily influenced the research community (Section 3.5.1), matchmakers for SAWSDL (Section 3.5.2), and further approaches that play an important role in the context of this thesis (Section 3.5.3).

3.5.1 Basic Approaches

Even though the number of matchmaking approaches is very large, there are some elementary contributions in (semantic) service matchmaking that have been picked up and enhanced by several other researchers.

To start with, the seminal work of Sycara et al. is, to the best of our knowledge, the first contribution that regards matchmaking in the context of SWS [241, 242]. The main focus of this work is on the agent capability description language "Language for Advertisement and Request for Knowledge Sharing (LARKS)" and not on a Web service standard. However, the approach is already called "service matchmaking", even though these services are actually provided and consumed by software agents. LARKS features several elements that can also be found in a similar way in (SA)WSDL or OWL-S with the declaration of variable types, input/output variables, input/output constraints (similar to preconditions and effects), and ontological descriptions of inputs/outputs as the most prominent features. The last-mentioned differs from the way semantic annotations have been conceived in this thesis – instead of determining, e.g., the type of an input semantically, the linguistic description of this input is semantically enriched.

Nevertheless, LARKS can be used in order to advertise and request agent capabilities in a similar way to the way this is done using (semantic) Web service standards. In the actual matchmaking, Sycara et al. make use of different "filters" [241]:

1. A *context* filter, which is used in order to determine if advertisement and request are from the same domain. This is done by determining the semantic similarity between textual domain descriptions.

2. A *profile* filter applies Term Frequency-Inverse Document Frequency (TF-IDF) on the entire agent specification.

3. A *similarity* filter is applied to single parts of the specification. Here, the semantic similarity of, e.g., pairs of input and output declarations is computed and afterwards assembled to an overall similarity.

4. The *signature* and *constraint* filters are applied together in order to determine plug-in relationships between the agents' signatures (input and output) and the corresponding (input and output) constraints.

The filters complement one another and provide different degrees of accuracy and computational costs. In LARKS, the following types of matching are defined [242]: (i) The *exact* match, where two descriptions are equal, (ii) the already mentioned *plug-in* match, which is essentially met if a service provides more than the functionalities necessary (or exactly those as the exact match is a special case of the plug-in match), (iii) the *relaxed* match that is met if the similarity value for the compared descriptions exceeds a predefined threshold. In contrast to the other matches, the relaxed match does not ensure any degree of compatibility but "only" defines that a certain degree of similarity is given. Regarding the abovementioned filters, the first three filters are used for relaxed matches while signature and constraint filters are applied for plug-in and exact matches.

Apart from creating a foundation for SWS matchmaking in general, three aspects of LARKS and the corresponding matchmaker should be explicitly mentioned: First, the usage of subsumption matching in order to determine relationships among concepts, second, the determination of plugin matches, and third, the hybrid approach that incorporates both implicit and explicit semantics. The work conducted by Sycara et al. has heavily influenced the SWS matchmaking research community. Most notably, the matchmaking approaches of Paolucci et al. and Klusch et al. presented in the following are based on LARKS [134, 136, 140, 201].

Within their work on the integration of DAML-S into UDDI-based service registries (cp. Section 4.5), Paolucci et al. also provide a matching approach, which is, to the best of our knowledge, the most-cited work in this research area [200, 201, 243]. The authors discuss a concept for matching of Web services based on DAML-S, a predecessor of the OWL-S standard. While with DAML-S several sophisticated ways to describe a Web service are introduced – most importantly effects and preconditions – the matching is restricted to inputs and outputs (the service signature).

Paolucci et al. define the four levels of *exact*, *plug in*, *subsume*, and *fail*. These DoMs are based on logic subsumption matching, i.e., the ancestry relationships between concepts in an ontology. As discussed in Section 3.1.1, a ranking of DoMs is defined. The authors also present the concept of *global DoM* for two services, which they define as the worst DoM found during matching. Thus, a minimum degree of compatibility regarding service inputs and outputs is achieved. One further restriction defined in this work is that every input in the service offer and every output in the service request needs to be matched by a corresponding element in the service request respectively offer. Another assumption is that if a service advertisement features a certain service output/input annotated with a semantic concept, *every* subclass of this concept is also offered.

The ranking of services is based on the assumption that a requester prefers a high global DoM for outputs more than a high global DoM for inputs. Hence, matching services are first sorted by the global output DoM. If these are identical for two or more services, the global input DoM is also taken into consideration. A further ranking of service advertisements is not arranged for [201].

The authors apply a *greedy* approach to the actual service components' matching. In a nutshell, this greedy approach takes the first semantic concept from a set (representing the inputs or outputs of a service request) and determines the corresponding concept with the highest DoM in the set of inputs (or outputs) of a service offer. Shortcomings of this approach have been discussed in Section 3.1.3.

While the matchmaking approach in the work of Paolucci et al. is more conceptual, an early implementation and enhancement of these ideas has been conducted by Li and Horrocks [164, 165]. In this work, services are represented by DL constructs. Hence, it is possible to conduct matching by DL inferencing. The authors present a way to model DAML-S-based services using DL notions; more precisely, this is limited to inputs and outputs, further elements are not regarded. Afterwards, it is possible to use *Racer DL* as reasoner which computes semantic matches between service offers and requests. As the name implies, Racer DL is suitable for semantic concepts defined in a description logic like $SHIQ(\mathbf{D})$ or DAML+OIL (which is a predecessor of OWL) [109, 110].

While Paolucci et al. define DoMs for single inputs and outputs, Li and Horrocks determine this by DoMs between service offer and service request represented by input and output parameters, i.e., a match is *exact* if the offer and request concept are equivalent, *plug in*, if the request is a subconcept of the offer, *subsume* if the request is a superconcept of the offer, *intersection* (not regarded by Paolucci et al.) if the intersection between request and offer is satisfiable, and *disjoint* otherwise. In DL, a logical intersection

of two concepts is *satisfiable* if there possibly exist individuals belonging to the resulting (intersection) concept. Again, DoMs are ordered in a discrete scale with exact as the best DoM and disjoint as the worst.

A distinct approach to service retrieval is proposed by Bernstein and Klein [28, 127]. Like LARKS, this approach is not aiming directly at Web services described, e.g., in WSDL, but at services in a more broader sense, i.e., software applications, software components, best practice repositories, and even individuals or organizations. The approach of Bernstein and Klein is based on process ontologies, hence, it is necessary to describe a service by a process model which is indexed into a process ontology. Queries are defined using the Process Query Language (PQL) which allows to define queries for entities, relationships, and attributes from the process ontology. In contrast to the subsumption matching proposed by Paolucci et al., PQL allows a more fine-grained definition of what a service needs to offer, e.g., that only tasks that are a specialization of a given task are valid. In the end, only these services that fulfill all clauses from the PQL query are defined as matches, i.e., the pattern given by the query needs to be satisfied. A more detailed discussion of PQL can be found in Section 4.5.

A comprehensive early approach has also been presented by Cardoso and Sheth [47]. Like the work by Verma et al. (see below), this work was part of the METEOR-S project. Cardoso and Sheth regard several aspects of service-based workflow composition, including semantic-based matchmaking of functionalities, non-functional requirements, and service interoperability. Regarding discovery of service functionalities, both syntax- and semantic-based techniques are deployed. Syntactic similarity measures are deployed for textual service names and service descriptions; semantic-based measures are used to determine the similarity of inputs and outputs. However, matching is primarily conducted for semantic integration, i.e., how the output of one service can be used as input for another service. In a follow-up work by Cardoso, service matchmaking as reckoned in this thesis is also regarded: Here, the similarity between concepts is determined based on the properties of inputs and outputs [45]. Again, different DoMs are proposed: *Equal*, *Specialization*, *Generalization*, and *Intersection/Disjoint*. As it was mentioned in Section 3.2.1, the ranking scheme of Cardoso differs from that of Paolucci et al. (in line with Bellur et al. [20]): Cardoso favors service offers that expect more generic input over those with a more specific input, as compared to the desired input in the service request. In case of outputs, a reversed ranking scheme is proposed. The actual similarity is a numerical value which is based on the number of properties in the service signatures of request and offer. As a major extension, Cardoso proposes matching without a common ontology commitment. For this purpose, a mapping between concept classes from different ontologies is performed and the geometric distance between the similarity of the domains of the concepts is computed. The mapping is based on syntactic similarity [45].

The approaches presented so far can be rated as the foundation for most of the matchmakers that followed. Of course, these approaches are extended, but in most matchmakers, several aspects can be found which make use of ideas first presented (cp. Sections 3.5.2 and 3.5.3). Regarding LOG4SWS.KOM and COV4SWS.KOM, especially the identification of subsumption relationships between concepts plays a very important role.

3.5.2 Matchmaking Approaches for SAWSDL and WSDL

To the best of our knowledge, the number of semantic matchmakers for SAWSDL-based Web services is quite low: 2009's S3 Contest featured four matchmakers for SAWSDL, including an earlier version of COV4SWS.KOM named COM4SWS [142]. In comparison, seven matchmakers for OWL-S were taking part in the contest. In the following, the contestants from 2009's S3 Contest as well as some other approaches to matchmaking for (SA)WSDL will be presented.

In their early work regarding semantic annotations for WSDL, Sivashanmugam et al. also present a basic discovery mechanism which is part of the METEOR-S framework [209, 231–233, 250]. Even though the work aims at a preliminary version of WSDL-S, it can also be assigned to SAWSDL [130]. Matching is performed for both functional and QoS requirements. Regarding functional matching, semantic matching is executed for operations, inputs, outputs, preconditions, and effects. As a distinctive feature, weights can be assigned to each of the five service elements – i.e., the service requester can rate to which degree each semantic part will be regarded in the overall service matching. The actual semantic matching value is a numerical value in the range 0 to 1. If the semantic concepts in service offer and request are identical, the semantic matching value is 1; if the concepts are not identical, a linear function is used to calculate the similarity value based on a *subClassOf* hierarchy [233].

Another early matchmaking approach for WSDL has been presented by Syeda-Mahmood et al. [244]. Here, the authors combine similarity values based on domain-independent and domain-specific ontologies. Matching is restricted to inputs and outputs. Regarding domain-independent semantics, service element names are captured, tokenized, further processed, and synonyms are detected using WordNet. Domain-specific semantics are based on WSDL-S *modelReferences* which are similar to the model references defined in SAWSDL. Inferencing is then conducted on the semantic annotations, resulting in one of the possible relationships *EquivalentClass* (0.0), *subClassOf* (0.5), *superClassOf* (0.5), and *RDFType* (0.0). Apart from the last-mentioned, these relationships can be traced back to the DoMs defined by Paolucci et al. Afterwards, a numerical value for the semantic relationships is determined (see brackets); this value is 1.0 if no relationship could be detected. The overall matching result is the maximum value of the domain-independent or -specific similarity values.

Even though not making use of explicitly defined semantics, the Web service search engine for WSDL *Woogle* provides some interesting aspects [73]. In this approach, matching is operations-centered as it is similarly applied in LOG4SWS.KOM and COV4SWS.KOM. However, as WSDL is missing semantic annotations, only implicit semantics as well as textual descriptions can be applied. Similar to semantic service matchmaking approaches, Woogle demands a complete WSDL description as service query by example. The exploitation of implicit semantics is based on input and output parameter names and their relationships. Parameters are clustered into semantically meaningful groups based on the assumption that "parameters tend to express the same concept if they occur together often" [73].

The matchmaker by Plebani and Pernici, URBE, utilizes linguistic as well as logic information and is applied to WSDL 1.1 [212]. The authors argue that Web service descriptions are usually derived from software components that implement the underlying functionality. In turn, this implies that the contained descriptions will follow some naming convention and thus constitute a meaningful source of information in matchmaking. In URBE, different service abstraction levels are taken into account, namely port types (WSDL 1.1's equivalent to interfaces in WSDL 2.0 [32, 59]), operations, and inputs/outputs.

The authors make use of two distinct similarity measures: First, the similarity of two service element's names is measured, based on domain-specific as well as on general word ontologies. Like in our fallback strategy, the names are tokenized before the similarity is calculated (on the set of terms derived from the names). For port types, only their names are taken into consideration for the similarity. For operations, the names of the operation and its parameters are taken into account for this similarity measure. For parameters, the value is combined with a second one, which assesses the associated XSD data type for a given pair of inputs or outputs. Therefore, a predefined similarity value is used, depending on the information loss that occurs if converting the two types.

After the similarity values have been computed, a maximization function is used in order to determine the assignment of elements from a request and offer. This assignment and maximization function is based on bipartite graph matching and is separately applied to sets of terms, sets of parameters, and a set of operations. The authors do not specify which concrete algorithm is applied. The overall similarity of a set is determined by the average of the assigned weights of edges in the bipartite graph. The overall similarity value for a service interface is the sum of the similarity values on the different service abstraction levels. This sum can be altered and tuned by determining weights for the levels.

As an extension, the authors also present a semantic similarity function, called URBE-S, which replaces the name similarity measure if semantic annotations are available. Provided with two concepts of the same type (i.e., classes or properties), this function measures the path length between them in the ontology. In contrast to other approaches, Plebani and Pernici distinguish between classes and properties. As it can be seen from Section 3.4, URBE(-S) is quite effective regarding recall and precision.

Extending the family of "MX"-matchmakers which has originally been defined for OWL-S (cp. Section 3.5.3), Klusch et al. provide their matchmaker SAWSDL-MX in different variants [132, 140, 141]. Their approach calculates three kinds of similarity, based on logic, textual information, and structure, and adaptively learns the optimal aggregation of those measures using a given set of services. Matching is employed on the level of interfaces using a bipartite graph matching algorithm, i.e., similarity values for all combinations of operations from a service request and offer are calculated. Afterwards, the best assignment is calculated.

Logic-based matching is conducted for operations and based on the subsumption DoMs by Paolucci et al. [201]. However, Klusch et al. reverse the ranking of more specific respectively more general classes. The DoMs are defined for whole operations and not for single parameters. However, a native

matching on operation (or interface) level is not intended. The DoM of operations is directly derived from their parameters, i.e., the matching degree of an operation is directly constrained by the minimal subsumption DoM found for its message parameters. Again, the matching of parameters from service requests and offers is determined by bipartite graph matching. An *exact* match is given if there exists a one-to-one mapping of perfectly matching input and output parameters; a *plug-in* match relaxes the exact match by allowing more general input parameters and output parameters that are direct child concepts of the requested concept; a *subsumes* match relaxes the former DoMs by allowing output parameters of the offer to be generally more specific than specified in the request; a *subsumed-by* match additionally allows output parameters to be more general. A *fail* match is detected if none of the aforementioned matches can be determined [132]. Syntax-based similarity values are used in order to refine matchmaking results and provide a sophisticated ranking of services. Therefore, the semantic concepts used to annotate input and output parameters are unfolded and result in one set of terms for *all* input parameters etc. A similarity value for the resulting sets of terms can then be determined using *SimPack* (see below).

Klusch et al. also introduce *structural* matching using the *WSDL Analyzer* tool [140]. WSDL Analyzer applies a recursive three-step approach: Operation sets are compared based on the structure of the respective input and output messages which in turn is based on the comparison of parameter data types. SAWSDL-MX is adaptive regarding the three similarity measures presented – using a Support Vector Machine (SVM), it learns the optimal aggregation of these measures. More precisely, the three measures are independently calculated; using a training set made up from some queries and their corresponding result sets, a SVM then calculates a non-linear aggregation function [141].

There are different variants of SAWSDL-MX: SAWSDL-M0 is a crisp-logic-based matchmaker which does not make use of syntax-based similarity measures; SAWSDL-MX1 combines the abovementioned logic- and syntax-based matching in order to further rank services at every logic-based DoM; SAWSDL-MX2 is the adaptive variant of SAWSDL-MX1. Furthermore, there is a variant of SAWSDL-M0 that uses WSDL Analyzer in order to further rank services (analog to SAWSDL-MX1) [140].

In [245], Tran et al. depart from the usual definition of subsumption-based DoMs and propose the application of more fine-grained matching results [245]. Additionally, the authors propose the usage of many-to-many mappings between parameters instead of the usual one-to-one mappings. Therefore, parts of a top-level parameter in a service request might be mapped to parts of another top-level parameter in a service offer. Matching is done on operation level. The authors define four "match degrees": *service input fulfillment* (SIF), *request input redundance* (RIR), *request output fulfillment* (ROF), and *service output redundance* (SOR). As the names imply, SIF and ROF define the fulfillment of service inputs and request outputs; RIR and SOR are the ratio of redundancy of request inputs and service outputs. A continuous scale ranging from 0 to 1 is used for all values. Furthermore, four "match levels" are defined: *precise*, *over*, *partial*, and *mismatch* which are specified on the match degrees; e.g., a match is defined precise if $SIF = ROF = 1$ and $RIR = SOR = 0$. Partial and mismatch are defined using customizable thresholds. Similarities are based on semantic annotations of upper elements and lexical checking for lower elements [245]. Unfortunately, the authors do not provide an evaluation of the matchmaking performance of their approach.

Kourtesis and Paraskakis also provide a matchmaker for SAWSDL in the *FUSION Semantic Registry*, which is presented in more detail in Section 4.5 [146]. This approach is based on ontological representation of service requests and advertisements in *Functional Profiles*. Matching is conducted on three distinct levels: boolean functionality-level matching, which is based on a high-level categorization of a service, DL-based message-level matching following the approach by Li and Horrocks mentioned above [164], and schema-level matching which is applied to determine a possible mapping between input and output data sets. Again, this mapping is based on the approach by Li and Horrocks [164]. Syntax-based or hybrid matchmaking is not regarded.

3.5.3 Further Approaches

Last but not least, we will present further matchmaking approaches which have got a relationship to the work at hand. As it was mentioned before, a large number of matchmakers exists – in 2008, Klusch et al. listed more than 35 matchmakers in his categorization of SWS matchmakers [130, 142]. Not all of the techniques used by these matchmakers are of interest for the work at hand, so the following

descriptions will be limited to those matchmakers that either have influenced the work on LOG4SWS.KOM and COV4SWS.KOM or had in our opinion a significant impact and visibility in the research community.

To start with, the work of Guo et al. and Bellur et al. have heavily influenced the matching in LOG4SWS.KOM and COV4SWS.KOM as presented in Section 3.2.1 [17–20, 97]. Guo et al. were the first to explicitly apply bipartite graph matching to service matchmaking. In their scenario, the parameters of service request and service offer constitute a bipartite graph, i.e., two distinct sets of nodes, where edges solely exist between the distinct sets, but not within them. Matching is limited to inputs and outputs; the applied service description language is OWL-S. Accordingly, the matching of services is transformed into a bipartite graph matching between their parameters. Edge weights are computed using three distinct and weighted similarity measures, which include linguistic similarity, class property similarity, and semantic concept similarity, based on logical reasoning [97]. Unfortunately, the description how these similarity measures are calculated remains quite superficial. The Hungarian algorithm is used to solve the bipartite matching problem. In contrast to most other approaches, the authors do not assume that all semantic concepts used are from one ontology but allow heterogeneous ontologies. In case two concepts are not contained in the same ontology, the derivation of similarity values based on the parameter names is defined. Here, a standard language ontology (in this case: WordNet [186]) comes into effect. Guo et al. do not use a discrete, but a continuous scale to describe the DoM between services, ranging from 0% to 100% and denoting the relative equivalence between two services [97]. However, it is not described how the numerical equivalents should be calibrated.

Bellur et al. introduce a major extension to the previously described work by Paolucci et al. [17, 20]. The authors demonstrate by a number of examples that the approach for semantic matchmaking of services, introduced by Paolucci et al., may deliver poor results in certain scenarios and encourage the overly generic annotation of Web services. As an alternative – in line with the aforementioned work by Guo et al. – the utilization of bipartite graphs in the matching of service capabilities is proposed, where parameters constitute the graph's nodes and edge weights are set based on the semantic DoM between the associated concepts. As discussed in Section 3.1.3, Bellur and Kulkarni show that the Hungarian algorithm for bipartite graph matching is capable to derive a global DoM [17]. Matchmaking is applied to inputs and outputs [17], an extension that incorporates preconditions and effects is also presented [18]; the service description language applied is OWL-S.

The authors rely on the discrete levels proposed by Paolucci et al. and the notion of a global DoM [17]. Interestingly, the concepts of plugin and subsumes as originally defined are inverted, i.e., if an output of a service request is a superclass of an output of a service offer, the corresponding DoM is a plugin etc. This can be traced back to the fact that Bellur et al. interpret semantic concepts in a different way and drop the assumption that if a concept is advertised, *all* its subconcepts will also be provided [18]. The discrete DoMs are converted into numerical values for bipartite matching purposes – the weights are based on the discrete DoM and the number of parameters in the request set [17]. If the whole service profile (cp. Section 2.2.3) is regarded, 14 DoMs are defined and ranked which combine parameters and preconditions/effects – here, parameter matching is crucial and preconditions/effects DoMs are used in order to further distinguish between parameter DoMs [18]. In a later work of the authors, an extension that enables a more fine-grained ranking of services concerning a service request is introduced [19].

The already-mentioned family of "MX" matchmakers has been published in several variants for SAWSDL (cp. Section 3.5.2), OWL-S (e.g., [131, 133, 136, 137, 139]), and WSML (e.g., [119, 120, 134, 138]). In fact, SAWSDL-MX is the latest variant of the matchmakers – hence, the following explanations primarily aim at aspects that differ between OWLS-MX and SAWSDL-MX; as WSMO/WSML have not been regarded in this thesis, WSMO-MX will not be examined.

Starting in 2005, OWLS-MX has been continuously enhanced to a hybrid adaptive matchmaker for OWL-S. In the following, the current version of OWLS-MX (as well as OWLS-MX2 and OWLS-MX3) as presented in [133, 139] will be examined. OWLS-MX is based on LARKS (cp. Section 3.5.1). The basic concept of OWLS-MX is the combination of logic-based reasoning and non-logic matchmaking techniques which rely on *implicit* semantics in service descriptions. Matchmaking is performed on service inputs and outputs. Semantically described inputs and outputs are unfolded, resulting in a set of index terms. To compare such sets different string similarity metrics are used [136]. Regarding logic-based reasoning, three different "filters" are computed: *Exact*, *Plug-In*, and *Subsumes* are defined concurrent with their counterparts in SAWSDL-MX.

There are two variants of the *Subsumed-by* filter: The first has been introduced in a latter variant of OWLS-MX, is also logic-based only and complies to its counterpart in SAWSDL-MX [139]. The second variant is hybrid and incorporates syntactic matching: Additionally to the logic-based variant, a syntax-based similarity threshold has to be met [136]. Last but not least, a *Nearest-neighbor match* filter is also applied. This filter is based on the abovementioned text similarity measurements and applied to both inputs and outputs [139]. Apart from the five filters, *Logic-based fail* and *Fail* are also possible results. The former means that a service offer fails to match with a service request regarding the four filters mentioned first. If none of the other results is detected, a *Fail* occurs [133].

OWLS-MX originally features five variants – the first variant OWLS-M0 applies only the semantic filters Exact, Plug-In, and Subsumes. OWLS-M1 to -M4 apply additionally different syntax-based similarity measures [136]. The latest version is OWLS-MX3 which incorporates the adaptive combination of different similarity measures for given queries [133]. Furthermore, [133] introduces "structural semantic matching" in OWLS-MX3 which is based on distances between concepts in an ontology. More precisely, this approach is based on the assumption that distances of rather specialized semantic concepts (i.e., concepts that can be found quite deep in a taxonomy) are more meaningful than distances between more generic concepts defined in the upper parts of an ontology/taxonomy. To the best of our knowledge, OWLS-MX3 is the only matchmaker which incorporates an approach from the field of semantic relatedness apart from COV4SWS.KOM. The structural matching also considers differing parameter numbers – if the number of outputs in a service offer or the number of inputs in a service request is smaller than in the counterpart, this has a negative effect on the overall structural similarity value [133].

The three similarity measures – logic-based DoMs, syntax-based, and (ontological) structure similarity – are combined as introduced for SAWSDL-MX2 [140]: A binary SVM-classifier is trained using a service test set and provides how the single values need to be combined [133].

The adaptive combination of different similarity measures is a well-known method which has been applied, e.g., for ontology mapping [75]. The first to propose the automatic adaptation of matchmakers to a particular domain or task have been Kiefer et al. [123, 124]. For their OWL-S-based matchmaker *iMatcher*, the authors make use of numerical similarity values and similarity functions from *SimPack* [29], e.g., extended Jaccard similarity and TF-IDF, which are integrated into SPARQL queries using Imprecise SPARQL (iSPARQL) (cp. Section 4.5) [123]. Similarities of semantic concepts in terms of subsumption relationships etc. are not directly regarded – instead, semantic concepts are unfolded and the similarity algorithms are applied to the names of the unfolded semantic concepts or a particular part of a resource description (i.e., the service name). The combination and weighting of different atomic similarity measures is automatically determined using machine learning approaches from Weka [256] and LibSVM [53], including a linear regression model [123].

As it was said before, the number of matchmakers is very large and not every approach has had an impact on LOG4SWS.KOM or COV4SWS.KOM. For example, the work of Jaeger et al. has been influenced by LARKS [241], where this approach is partly adapted and enhanced [114, 115]. An interesting approach where service signature and specification are integrated in order to combine matchmaking of inputs/outputs and preconditions/effects has been introduced by König-Ries et al. [128, 155]. As a distinctive feature, this matchmaker combines service matchmaking and service composition. Similar to the work of Li and Horrocks, Di Noia et al. and Benatallah et al. make use of DL [21, 68]. Their approach is based on non-monotonic DL reasoning, making it possible to achieve a fine-grained ranking. Other DL-based approaches are presented by Grimm et al. where the authors make use of local closed-world DL reasoning [91, 92], and d'Amato et al. where services are clustered in order to make discovery very efficient [65]. Other approaches make use of clustering in order to downsize the number of service offers that need to be regarded [2, 193, 229].

3.5.4 Overview

As it can be seen from the last sections, the range of possible techniques is quite broad and ranges from logic-based methods to IR-based similarity values. This diversity of approaches is an explanation for the multitude of matchmakers that have been proposed for different service standards in recent years. However, there are some approaches that can be deemed as seminal, e.g., [123, 136, 164, 201, 241]. Regarding SAWSDL, the work by Plebani and Pernici and Klusch et al. is without question the most

related work to LOG4SWS.KOM and COV4SWS.KOM. Hence, these approaches have been chosen as benchmarks in the evaluation (cp. Section 3.4).

Most approaches make use of logic-based similarity measurement between concepts in an ontology. Apart from the path length, alternative means to derive the semantic relatedness of concepts are not applied. Even though a lot of different approaches have been proposed, the missing integration of all service abstraction levels is noticeable. Most matchmakers presented are restricted to the service signature, in some cases the whole service profile (including preconditions and effects) is regarded. This might be traced back to the fact that still extensive, heterogeneous ontologies are missing which are needed to semantically describe service components [77, 103]. Therefore, we integrate fallback strategies in LOG4SWS.KOM and COV4SWS.KOM, which are usually missing (with the work of Syeda-Mahmood et al. and Plebani and Pernici being notable exceptions).

The integration of different service abstraction levels further raises the question on how to weight the different levels (cp. Section 3.2.1). Here, the automatic adaptation as conducted in COV4SWS.KOM seems to be a good choice. In comparison, the adaptive matchmakers presented by Kiefer et al. and Klusch et al. are adaptable regarding the combination of different similarity measures. As a last aspect, it should be noted that runtime performance is usually not regarded explicitly. In LOG4SWS.KOM and COV4SWS.KOM we make use of caches and ATSM in order to boost the runtime performance.

3.6 Conclusions

Service matchmaking based on semantic information is reckoned by a very agile research community, with a large number of different approaches proposed in recent years. A lot of experimentation is conducted regarding which data should be included in the matchmaking process, applicable similarity measurements, and how the resulting values should be combined and evaluated. The matchmakers presented in this chapter complement and pursue the already existing work by combining accepted techniques and new ideas resulting in very good results regarding IR metrics like recall and precision. To the best of our knowledge, LOG4SWS.KOM and COV4SWS.KOM provide the best AP values of any matchmaker for SAWSDL so far; this assumption is backed up by the other IR metrics regarded.

While LOG4SWS.KOM and COV4SWS.KOM make use of similar respectively identical methods regarding the actual matching of service components, the fallback strategy, and caching mechanisms, especially the ontology-based similarity measurement of service components differs – LOG4SWS.KOM applies an enhanced variant of the classic DoMs by Paolucci et al., while COV4SWS.KOM makes use of similarity metrics from the field of semantic relatedness. Matchmaking is aiming at service similarity, i.e., the identification of similar services to a given service. The found services might have to be adapted to the service requester's needs.

LOG4SWS.KOM and COV4SWS.KOM combine all service abstraction levels available in SAWSDL and are not restricted to the service signature. In the case of COV4SWS.KOM, an automatic adaptation to different degrees of usefulness of the descriptions of different service components is introduced. This feature had to be omitted in LOG4SWS.KOM as an automatic adaptation is here already done for the numerical equivalents of usually distinct DoMs. Per se, it would be beneficial to have semantic descriptions on every service abstraction level, however, if certain service components miss semantic annotations, a syntax-based fallback strategy can be applied.

Regarding the hypotheses presented at the beginning of this chapter, we can make the following statements:

H1: The automatic derivation of numerical equivalents of subsumption DoMs as conducted in LOG4SWS.KOM has led to an improvement of matchmaking results as can be seen in Table 3.4. In 2 out of 3 cases, all IR evaluation metrics are higher than for the manually derived numerical equivalents, even though these have been already optimized for SAWSDL-TC (cp. Appendix B.2.3). For the third case, the RP and P(10) values are slightly improved by the automatic determination of numerical equivalents, while AP and P(5) values slightly decrease. Thus, we can deduct that the automatic derivation of numerical equivalents for subsumption DoMs, which relates to the ranking and treatment of more generic and specific semantic concepts (cp. Section 3.1.2), leads to an improvement of matchmaking results.

H2: The computation of similarity metrics on all service abstraction levels instead of only considering the service signature leads to an improvement of matchmaking results. However, the weighting is important: Regarding the applied test data set, levels which feature only syntactic descriptions should be integrated but weighted to a lesser extent than those levels which feature semantic descriptions of service components. However, the actual ideal weighting depends on the similarity metrics applied – in case of COV4SWS.KOM, manual weighting performs better than the automatically derived ones, except for the similarity metric proposed by Resnik.

H3: Even though the evaluation results for COV4SWS.KOM are usually lower than those of LOG4SWS.KOM, similarity metrics from the field of relatedness measurement of semantic concepts in ontologies are still performing well, especially if compared to other matchmakers (cp. Table 3.6). Most importantly, the automatic weighting of similarity values from different service abstraction levels in combination with the metric proposed by Resnik leads to the best evaluation result regarding AP for any SAWSDL matchmaker, so far.

Of course, there are possible extensions of the work conducted: To start with, it would be interesting to integrate preconditions and effects into the matchmaking process of LOG4SWS.KOM and COV4SWS.KOM. While the number of services observed in our evaluation and the competitive runtime performance of our matchmakers does not make it necessary to apply more sophisticated ways to downsize the search space, this could be necessary if following the vision of an "Internet of Services" where billions of services are available [72, 197]. Here, clustering or a top-down approach, which eliminates services not meeting certain requirements on interface and operation levels, could be integrated. Furthermore, it might be suitable to make use of alternative similarity measures or heuristics in order to detect similarities between service components. The same applies to matching algorithm – instead of the Hungarian algorithm applied in this thesis, another method might be more efficient if this is necessary. There might be further ways to enhance the retrieval performance of the matchmakers at hand – hence, we label the matchmakers presented in this thesis with LOG4SWS.KOM Version 1.0 respectively COV4SWS.KOM.

As a last aspect, most matchmakers neglect non-functional aspects like QoS which is also done in LOG4SWS.KOM and COV4SWS.KOM. However, an integration of the work of Berbner and Eckert would be straightforward and could easily be evaluated using an appropriate test collection [23, 74]. This could be done incorporating a semantic-based description of QoS characteristics as well as countermeasures if QoS requirements of a requester cannot be met [217, 223].

4 Query Formalisms for Semantic Web Services

In this chapter, the formulation of queries for service discovery is examined. Usually, approaches to syntactic service discovery rely on keyword search or category browsing, while queries aiming at SWS are mostly encoded using the same formalism as applied for the description of services, i.e., follow a "query by example" approach. This makes it necessary to define a preferably complete description of the service capabilities desired, i.e., a service profile in OWL-S or a service interface in SAWSDL. As a result, query formats are mostly incompatible regarding different Web service standards. Further deficiencies of current query formats are the missing integration of syntactic and semantic search and the absent consideration of ranges of values.

The following hypotheses are applied in order to determine the outcome of the work conducted in this chapter. These hypotheses will be discussed in Section 4.6.

H4: Current query formalisms for SWS do not provide the necessary means to define requirements in a user-friendly, flexible, and precise way. Hence, it is necessary to deploy alternative means to formulate service requests. Already existing query languages could be a foundation for this, even if they have not been initially developed for application in SWS discovery.

H5: A unified query formalism for SWS should be independent from any particular registry or Web service specification. Thus, it is necessary to define a (unified) query language and corresponding service data and query models, which are per se independent from but nevertheless applicable to different standards.

This chapter is structured as follows: First, shortcomings of current approaches to query formulation for (semantic) Web services are identified (Section 4.1). Based thereupon, requirements towards service query languages will be specified (Section 4.2). Following this, two distinct approaches to query languages for SWS will be described, which aim at different goals: The approach described in Section 4.3 has been the first approach designed and implemented in the context of this thesis. It aims at providing a lightweight integration of SPARQL-based queries for SAWSDL-based Web service descriptions into UDDI. In contrast, the second solution (presented in Section 4.4) provides a more extensive approach to enhance a service registry by alternative query languages. Therefore, an abstract query model is proposed. Based thereupon, SWS2QL, an extension of SPARQL, aiming at SWS retrieval, is conceptualized. As proof of concept implementation, an ebXML-based service registry is enhanced. Before the chapter closes with a summary, Section 4.5 gives an overview on related work in this field of research.

4.1 Problem Statement

Regardless if following the visions of service marketplaces or the "Internet of Services" [49, 197, 203] or considering a relatively manageable number of services in an intra-organizational scenario, the discovery of services is one of the core success factors for the invocation of services. Usually, service registries offer a discovery interface a service requester can use in order to query for a service (cp. Section 2.1.3).

The actual discovery is the process of locating a particular service from a set of services, which is capable to do a certain task in the most effective way [247]. For service discovery, two major aspects play an important role: First, it is necessary to provide a matchmaking algorithm (cp. Chapter 3). Second, the input for the matchmaker is given by the service request, i.e., a formulation of the requirements a service has to fulfill (cp. Section 2.3.1), and the service offers. The precise requirements vary on the basis of a user's intention and can comprise functional, non-functional, and technical aspects.

Based on the different approaches to query formulation presented in Section 2.3.1, we define three major query formalisms. These formalisms combine the way queries are expressed as well as their intention:

Keyword-based service requests are the typical query formulation for syntactic Web services; this includes wildcard- and table-based queries. Given a keyword as input, the matchmaker finds all instances

in which this word occurs. Keyword-based matchmaking is based on standard IR methods, e.g., term matching: The terms from the request are matched with keywords found in a service advertisement. Explicit semantic information is not regarded.

Query by example-based requests are widely adopted in research approaches which primarily aim on the deployment of new matchmaking algorithms (cp. Section 3). The basic idea is that the service request is formulated as a model instance, e.g., a WSDL-based service description. This model instance is deemed to be a "perfect service", i.e., the perfect answer to the service request. While this is a very rich request format, which can incorporate both syntactic and semantic information as well as the relationship between the single service components, there are a couple of shortcomings elaborated in the following.

Query language-based service requests make use of a structured query syntax, e.g., a query language like SQL or SPARQL. This makes it necessary to define some kind of service model as foundation for query formulation. Query language-based service requests do not directly describe a service as a model instance. Instead, parts of a virtual service model instance are *described* in terms of a well-defined query language. Service models provide the information and relationships required for creating a query language-based request. Query-based service requests can refer to both syntactic and semantic information.

Today, Web service portals like `www.seekda.com` primarily rely on keyword-based search which is not surprising as the listed services are mostly syntactically described [13, 161]. The same applies to state-of-the-art service registry standards like UDDI [62] and ebXML [86]. In Section 2.3.1, the several drawbacks of keyword-based retrieval have been discussed. As a result of these shortcomings, several approaches have already integrated semantic technologies into service registries, primarily UDDI (e.g., [145, 200]). The most common approaches are built on top of syntactic Web service descriptions, require different execution environments and do not make use of potentially already existing (syntax-based) query formats (cp. Section 4.5). Alternatively, it is possible to describe a service model instance, i.e., to follow the service description-based approach. This is a common assumption if evaluating matchmaking results – for this reason, the matchmaking results presented in Chapter 3 are also based on this query formalism.

Even though service queries by example allow to formulate requirements towards a service following both, subsumption-based retrieval as well as state-based retrieval, this approach possesses several drawbacks:

Integration of syntactic search: Service descriptions do not allow the explicit definition of syntactic information. As it has been shown in Chapter 3, it might be the case that a matchmaking algorithm takes syntactic information into account. However, this cannot be directly controlled by the requester if using a service description-based query.

Missing range: A service description does not recognize the definition of ranges for complete services and single elements. Thus, it is not possible to define, e.g., that only those services which *exactly* meet the request should be returned by a matchmaker. Again, a matchmaking algorithm could define that only certain ranges are included in the result set of a query. However, it is not possible to set according parameters in the service query if using the service description-based approach.

Incompatible query formats: If a service request is formulated in a certain Web service formalism, it is not possible to discover services which offer an interface in another Web service standard. This forces users to formulate different queries if a service registry is able to manage more than one Web service formalism and applies even to different versions of the same standard, e.g., WSDL 1.1 and 2.0.

Lack of knowledge: Service requesters are not necessarily familiar with one or more service standards. However, requesters might be familiar with a query formalism which is, e.g., already used in a particular infrastructure or domain.

Usability: Neither the fact that a customization or parameterization of queries has to be done at the matchmaker-level nor the need to formulate queries for different Web service standards in different formats is very user-friendly. It is quite likely that the usage of semantic information in service

requests will only be accepted by users, if it is possible to standardize and enhance Web service queries.

In order to search for services based on semantic information, the user needs to be able to define semantics in an intuitive and, if possible, uniform way, i.e., the request formulation should be independent from the service formalism(s) supported by the registry. From the user's point of view, the semantic discovery functionality should be a smooth extension of the current discovery functionality. The realization of such an integrated discovery should combine the available semantic discovery mechanisms with the traditional discovery mechanisms in a way which leverages their corresponding advantages. The abovementioned shortcomings show that current query formats for SWS miss features which would improve the formulation of service requests. Preferably, a unified query format for SWS should be offered by service registries. In the next section, the requirements such a language has to fulfill will be defined.

4.2 Specification of Requirements

Filling the gap between state-of-the-art service registries like UDDI and ebXML and fulfilling the goals of SWS and semantic discovery as, e.g., the facilitation of automated service discovery, composition, and execution of services starts with an integration of semantic service discovery in standard registries. This implies the integration of semantic information into a service registry, which has been suggested by many researchers as will be presented in Section 4.5. Furthermore, it is necessary to provide a query language, which is able to overcome the drawbacks of the usual request formats presented in the last section.

Requirements towards a query language can be divided into platform-dependent and -independent characteristics. For service discovery, the service registry standard used is the most constraining factor of a platform as it has to be altered or enhanced in order to facilitate a particular query formalism.

Based on the shortcomings of the keyword- and service-based approaches highlighted in the last section, the following generic requirements towards a query language for SWS have been identified:

Combination of syntactic and semantic search: A query language should integrate semantic as well as syntactic information. Furthermore, it should be possible to combine syntactic and semantic information within a single query.

Reuse: If a query language is already common to potential users, the chance to get it accepted will be likely higher. If multiple registry standards have to be supported, an appropriate mapping of the global query language to the specific registry query syntax should be provided.

Ranges: It should be possible to define a threshold of similarity a service offer needs to fulfill in order to be included in the result set.

As it was mentioned above, some of these requirements are currently reckoned in matchmaking algorithms, e.g., the usage of syntactic data or the definition of ranges or similarity thresholds (cp. Section 3.5). However, the actual handling of syntactic data or ranges in matchmaking is usually not explicitly regarded but used as a complement for semantic-based matchmaking. Hence, the usage of syntactic data and thresholds is not visible to the service requester and cannot be directly controlled. In fact, such a control would make it necessary to offer the requester different setting properties of a matchmaking algorithm, which need to be manipulated apart from the actual query. The definition of further search constraints within a query is a much more intuitive approach.

As it was presented in Chapter 3, most matchmakers rely on an example model instance as service request, i.e., follow the query by example formalism mentioned in the last section. Hence, it is necessary to alter or customize a matchmaker if permitting further query formats. This could be done by allowing a query processing engine to be responsive to different query formats and automatically choose or configure matchmakers.

Apart from the more generic requirements mentioned above, there are some requirements, which aim on the independence from concrete standards and need also to be regarded:

Flexible handling of different matchmakers: It should be possible to "manually" chose a matchmaker. If this is not done, the query processor needs to recognize appropriate matchmakers based on the actual query content.

Unified query syntax: The query syntax should be the same for all Web service standards addressed by the corresponding service registry.

Handling of different registry standards: A unified query syntax should be applicable to different registry standards.

Of course, there is some kind of trade-off between the different requirements. For example, the parameterization of different matchmakers might conflict with the reuse of an already existent query language, which might not necessarily support such a customization or selection of matchmakers.

In the following, we will show the integration of service query languages into two different service registry standards. First, the usage of SPARQL in a UDDI-based service registry will be presented. Afterwards, the integration of an abstract query model for services in ebXML Registry and the usage of the SPARQL-based SWS2QL show how an advanced query interface can be integrated in this service registry standard. In both cases, we will propose a model on how to integrate semantic information into the particular registry/repository standard, define corresponding queries, and integrate matchmakers. However, while the work presented in Section 4.3 is rather lightweight and only portable to some degree, the second approach presents a more conceptual approach, which is easily adaptable to work with different SWS formalisms and registry standards.

4.3 Integrating Semantic Discovery in UDDI with SPARQL

The first solution presented in this chapter is motivated by two practical reasons: First of all, UDDI is the de facto service registry standard applied in Web service research (cp. Section 4.5). Second, SPARQL is the standard query language for the Semantic Web [26]. Hence, it seems obvious to examine the applicability of SPARQL for SWS in combination with UDDI.

Per se, UDDI offers only a very simple keyword-based search interface (cp. Appendix A.2.1): Keyword-based search can be applied to different elements of the service representation in UDDI, namely the description of business services, business entities, or technical models. In addition, several approaches exist which integrate service description-based matchmaking into UDDI (cp. Section 4.5). In contrast, the usage of structured query languages in UDDI has not been regarded so far.

The remainder of this section is structured as follows: In the next section, we will motivate the usage of SPARQL as query language for UDDI. Afterwards, we will define a corresponding integration approach. In Section 4.3.3, the publication of SAWSDL-based service descriptions in UDDI will be examined. Section 4.3.4 presents the enhancement of UDDI by a SPARQL-based query interface. An overview of the implementation can be found in Section 4.3.5.

4.3.1 SPARQL as Query Language for UDDI

SPARQL is a query language for RDF, which possesses an SQL-similar syntax and represents query results as tables, boolean values, or RDF graphs (cp. Appendix A.4). Regarding the work at hand, SPARQL has been selected for several reasons. First of all, SPARQL is the major query language of the Semantic Web as depicted in Figure 2.5. Second, within SPARQL, query statements are expressed in terms of triples made up from a subject, object, and predicate (cp. Appendices A.3 and A.4). Thus, SPARQL represents a very intuitive approach, since the query statements are very close to shortened natural language-based expressions. Several enhancements of SPARQL for specific purposes have already been proposed as presented by, e.g., Bry et al. [39], showing the flexibility and adaptability of SPARQL towards different application areas: Although, SPARQL is intended to be executed against RDF-based data, it may also be applied as a query *description* language, which is the case within the work at hand.

The usage of SPARQL as a query language for SWS has been proposed especially in connection with OWL-S, e.g., [123, 221]. As OWL-S is OWL-based (which in turn is RDF-based; cp. Appendix A.3), the application of SPARQL as a query language suggests itself. But SPARQL has also been applied as query language for SAWSDL [113]. A general introduction to SPARQL can be found in Appendix A.4.

4.3.2 Integration Approach

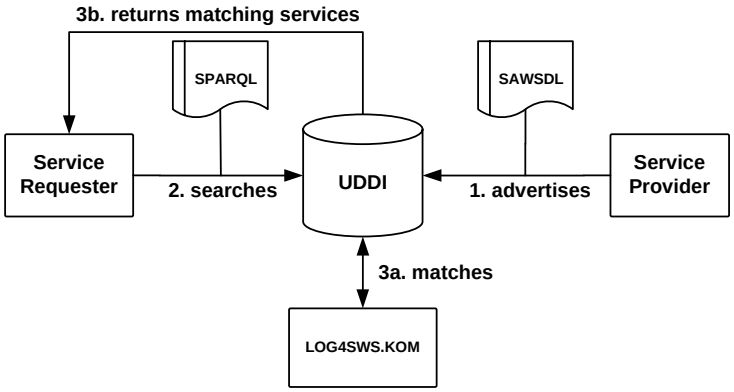


Figure 4.1: Service Discovery Scenario with UDDI

The enhancement of UDDI by further query formalisms requires different steps which – after the general query approach has been identified – are primarily predetermined by the registry standard or rather its implementation. Hence, it is necessary to follow a technology-driven approach. Figure 4.1, which applies Figure 1.2 to UDDI, SAWSDL, LOG4SWS.KOM, and SPARQL, gives an overview of the different steps in the publication and discovery/invocation process applied in this section. There are three main steps necessary in order to augment UDDI with SPARQL-based queries for SAWSDL:

Integration of SWS Formalisms: While the integration of SAWSDL- or WSDL-based service descriptions has been examined by, e.g., Sivashanmugam et al. or Kourtesis and Paraskakis [145, 232], the application of SPARQL needs further extensions. As SPARQL is a query language for RDF, its application to SAWSDL (which is XML- but not RDF-based) implies that a mapping between SAWSDL and RDF *or* the SPARQL query and an intermediate query format which can be directly applied to SAWSDL, has to be performed. In this solution, the first alternative has been used due to the fact that a mapping from SAWSDL to RDF is defined in the official W3C recommendation for SAWSDL [79], while there exists no official mapping from SPARQL to WSDL or an official query language for this Web service formalism. The integration of a new SWS formalism (here: RDF) requires the definition of a publication process for this new formalism.

Search Interface: UDDI users commit requests to a search interface. In the work at hand, it is necessary to design and implement an interface for SPARQL-based service requests.

Query Processing: After a request has been specified and submitted, it needs to be processed. Therefore, it is necessary to augment the search component of UDDI by adding matchmaking facilities which can handle the input format (here: SPARQL) and the applied service formalism (here: SAWSDL and RDF). As matching engine, LOG4SWS.KOM is adopted but enhanced in order to allow for ranges by specifying individual similarity thresholds. In order to keep the description simple, we apply a logic-based, service signature-only variant of LOG4SWS.KOM, i.e., Variant 1c (cp. Section 3.4.1).

4.3.3 Publishing SAWSDL-based Services in UDDI

In the following, the enhancement of UDDI necessary to reference SAWSDL- as well as RDF-based service descriptions, will be presented. This process is made up from the following steps:

1. Publication of RDF-based Web service representations using technical Models (tModels)
2. Integration of mappings from SAWSDL to RDF
3. Publication of tModels
4. Update/Deletion of tModels

As will be presented in Section 4.3.5, the implementation was conducted based on the open source software *Apache jUDDI*[1]. At the time of implementation, the version of jUDDI complying with UDDI Version 3.0 was not yet stable. Hence, the following assertions refer to a UDDI Version 2.0-based registry; however, the resulting differences are manageable. For differences between UDDI Versions 2.0 and 3.0 as well as a general introduction to UDDI, we refer to Appendix A.2.1.

Enhancement of UDDI tModels

The integration of Web services in UDDI can be done by either mapping single elements from a Web service description (like a SAWSDL or OWL-S file) to the UDDI data model or referencing the service descriptions [63]. Both approaches have their individual advantages and disadvantages, which have been discussed by Colgrave and Januszewski [63]. For example, storing all information within the UDDI makes it possible to find services even if the Web service (respectively the associated WSDL) is not available. Otherwise, the application of an SPARQL query is much easier if the RDF representation of a Web service description is directly available and does not have to be gathered from different tModels. We decided not to include the complete service description in UDDI but to reference the RDF- and SAWSDL/WSDL-based service descriptions for two reasons: First, the integration of WSDL elements as tModels makes the usage of extensibility elements in WSDL difficult, as those are usually not represented in the WSDL to UDDI mapping [63]. Second, the implementation efforts are lower. This design decision does not affect the information available and therefore usable in service discovery.

In UDDI, services are published by defining a `businessEntity` object, constituting a `businessService` object, and linking it to the business entity (cp. Appendix A.2.1). Finally, the service provider needs to create a `tModel` representing the WSDL. If not integrating the WSDL in UDDI, this is done by referencing the Uniform Resource Locator (URL) of the WSDL. In the work at hand, the course of action is exactly the same. However, after the WSDL has been registered in the UDDI a corresponding RDF file will be created and an RDF tModel will be stored in the UDDI (see below). Service requests specified in SPARQL will be later applied to the RDF files referenced by the tModels in the UDDI. The mapping from SAWSDL to RDF is presented in the next section.

To distinguish between tModels referencing WSDL files and those referencing RDF files, the former are called “WSDL tModels”, while the latter are called “RDF tModels”. In order to link the WSDL description and the RDF description, the following tModels are added to the standard tModels (*RDF tModels*, *RDF Reference tModel*, *RDF Categorization tModel*) or have been enhanced (*WSDL tModels*):

RDF tModels contain an URI pointing to the SAWSDL-based RDF file representing the WSDL-based service description.

RDF Reference tModel is used in a `keyedReference` to link a WSDL tModel with an RDF tModel.

RDF Categorization tModel is used in order to categorize RDF tModels. The usage as a categorization tModel is indicated by a Universally Unique IDentifier (UUID) pointing to this type’s tModel.

WSDL tModels link themselves to the corresponding RDF tModels by a `keyedReference` which is automatically constituted during the publication of a WSDL.

The detailed tModel descriptions and corresponding examples can be found in the Appendix D.1. As it can be seen, this enhancements are quite lightweight and do not require extensive modifications of the UDDI structure. Using the new tModels, a WSDL can now be published in the UDDI and RDF files can be created.

Integrating a Mapping from SAWSDL to RDF

The mapping from SAWSDL to RDF is achieved through a component called *WS2RDF.KOM* and based on the official mapping released as a note by the W3C Web Service Description Working Group [143]. Likewise, an official mapping is also included in W3C’s definition of SAWSDL, which itself is an extension

[1] `http://ws.apache.org/juddi/`

to WSDL [79]. In line with those publications, Kopecký provided an experimental Java implementation for automated mapping of SAWSDL documents to RDF, based on the *Woden4SAWSDL* framework. However, the recommendations – and hence Kopecký's implementation – do not include the mapping of XSD type definitions to RDF. Here, it was necessary to enhance the original mapping as presented in Appendix A.1.2.

Apart from making use of Kopecký's implementation, WS2RDF.KOM offers additional functionalities like a mapping from WSDL 1.1 to 2.0 based on an Extensible Stylesheet Language Transformation (XSLT) stylesheet provided by W3C. This feature makes it possible to apply SPARQL queries not only to WSDL 2.0-based service descriptions (which are in the focus of this thesis) but also to WSDL 1.1-based services, as presented in Appendix A.1.2.

In order to keep the publication process as simple as possible, the mapping from SAWSDL to RDF and the integration of this data is hidden from a service provider who wants to register a service in the registry.

Publication of a tModel

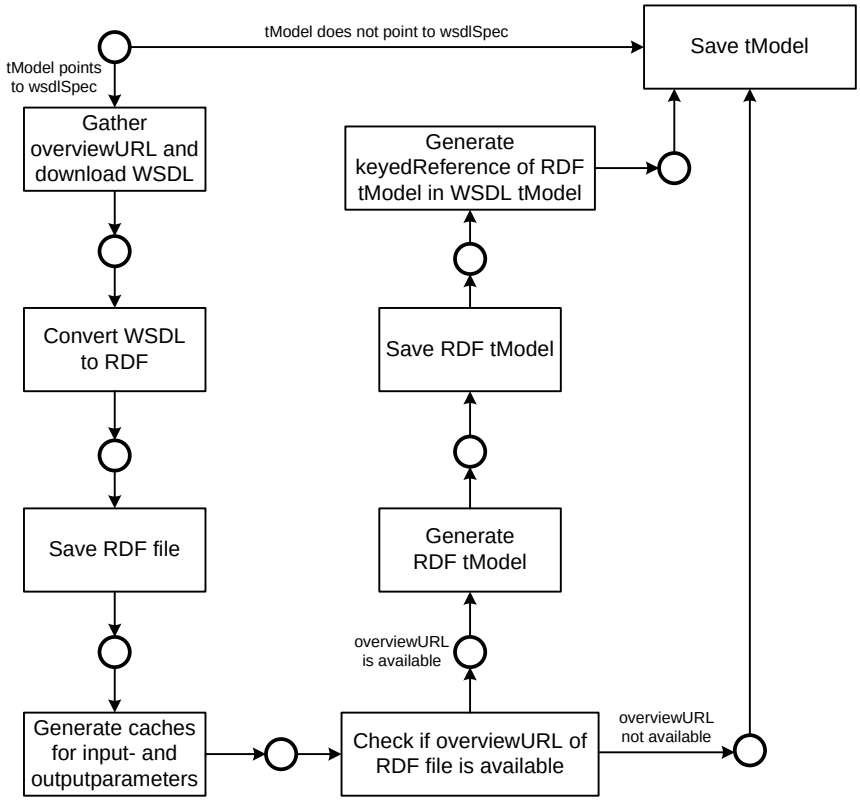


Figure 4.2: Enhanced Publishing Mechanism for UDDI

As it has been described before, a service description is published by adding a tModel to a UDDI registry. In order to reference the RDF mapping mentioned above, it is necessary to enhance the publication process as it can be seen in Figure 4.2. The enhanced publication process consists of the following steps; it is assumed that a tModel has already been created beforehand:

1. If a new tModel is added to the UDDI registry, its `keyedReferences` are checked whether referencing the `uddi-org:types` tModel and featuring the value `wsdlSpec`. The purpose of this step is to check if the tModel is addressing a WSDL.

2. If this is the case, the WSDL tModel is checked for the WSDL's overview URL and the WSDL file is downloaded. Else, the publication process ends and the non-WSDL tModel is stored inside the UDDI.

3. The WSDL file is converted to RDF using the WS2RDF.KOM component.

4. The RDF file is stored using the naming convention `nameOfWSDLTmodel.rdf`; afterwards, the downloaded WSDL file is deleted.

5. In order to speed up the actual discovery process, caching with regard to the referenced semantic concepts is carried out. This step is explained separately (see below).

6. After the cache files have been created, an `overviewURL` referencing the new RDF file will be returned. If it was not possible to create the RDF and cache files, the `overviewURL` will be an empty string and the tModel from Step 1 will be stored in the UDDI without any changes. Else, the RDF tModel containing the `overviewURL` will be generated.

7. In the last step, the WSDL tModel will be extended by a `keyedReference` referencing the RDF tModel (cp. Listing D.3).

Afterwards, the service representation needed for a SPARQL query is available in the UDDI.

If a WSDL is altered, its RDF representation needs to be updated. This process is actually achieved by adding a WSDL tModel to the UDDI possessing the same key. In such a case, the old WSDL tModel is not immediately deleted but scanned for possible keys referencing an RDF tModel. Afterwards, the RDF and cache files as well as the RDF tModel are deleted. Furthermore, the old WSDL tModel is removed from the service registry and the new tModel needs to be checked for keys referencing RDF tModels in order to avoid inconsistencies. Now, the publication process as presented above can be carried out. If a service is supposed to be deleted from the registry, the process is the same except for final steps, i.e., checking if an RDF tModel is already referenced in the new WSDL tModel and carrying out the publication process.

Caching

As it was mentioned before, the matchmaker integrated in this implementation is based on LOG4SWS.KOM. A preliminary reasoning during publication-time has been proposed by a number of researchers, e.g., Kourtesis and Paraskakis and Srinivasan et al. (cp. Section 4.5), and is also intended by the caching mechanisms presented in Chapter 3. The aim is to speed up the actual discovery process as costly subsumption reasoning between semantic concepts does not have to be carried out during the matchmaking runtime. In the section at hand, caching is introduced for subsumption-based matchmaking. However, similar caching mechanisms could be easily integrated for other matchmaking approaches, if necessary.

Before the actual reasoning can be carried out, it is necessary to identify semantic concepts in the Web service descriptions. Here, we apply SPARQL queries, which are presented in Appendix D.1.2. After the semantic concepts have been identified, reasoning is conducted using the *Pellet* reasoner [230]. The reasoning results are stored in separate cache files for different service components, i.e., separate caches exist for input and output parameters.

4.3.4 Integrating SPARQL into UDDI

After the RDF file describing a Web service can be referenced from within the UDDI, it is necessary to implement an interface which is able to process SPARQL queries and return a number of services matching the query. In the work at hand, SPARQL queries are formulated as strings and the corresponding result set is a number of `tModelKeys` represented as an XML-based document, which are sorted in a descending order based on the matchmaking result. A Web browser is used to submit service requests and to display the query results.

In the following, the actual query process, the formulation of SPARQL queries, and some extensions to the SPARQL query language and its processing, which add support for the definition of ranges based on a threshold and for simplifying the queries, will be presented.

Web Service Queries with SPARQL

WS2RDF.KOM constructs RDF files representing WSDL 1.1 and 2.0-based Web services following a particular pattern. Thus, SPARQL queries can be composed, which follow the pattern presented in the RDF mapping (cp. Appendix A.1.2).

SPARQL queries are formulated using *concept blocks*. Multiple concept blocks can be combined to an `ASK` query as depicted in the examples in Appendix D.1.3. In order to define a threshold of similarity, it was necessary to enhance standard SPARQL queries. This is done regarding the minimum subsumption DoM allowed for a certain service component. The actual definition is done by adding the string ".SIMILARITY." and the minimum DoM to the URI of the referenced semantic concept. In our implementation, the following DoM as defined in Section 3.2.1 can be specified: *exact*, *super*, *sub*, and *fail*. *fail* is the default value if no DoM has been defined. By adding a threshold, it is possible to specify more precise queries in comparison to the usually adopted, service description-based approach. In the work at hand, we make use of an enhanced implementation of LOG4SWS.KOM. For details regarding the actual matchmaker, we refer to Sections 3.2 and 3.4. However, if a matchmaker addresses other DoMs, these could also be applied as long as the matchmaker is able to process minimum DoMs.

SPARQL-based Search Process for UDDI

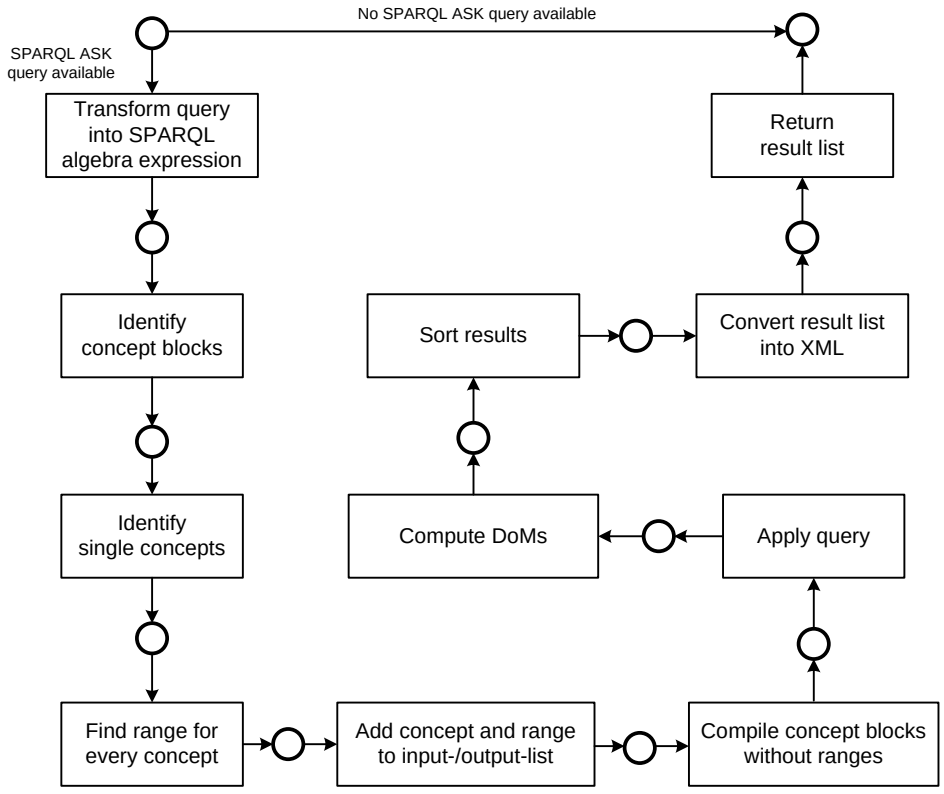


Figure 4.3: SPARQL Search Mechanism for UDDI

After the string representing a SPARQL query has been defined, the query process is started. The whole handling of SPARQL queries is done using the *Jena Semantic Web Framework* [51], which facilitates the handling of SPARQL queries as Java objects and also integrates a SPARQL processor (cp. Section 4.3.5). The following list shows the query process which is also depicted in Figure 4.3. An example query as well as its corresponding regular expressions are depicted in Appendix D.1.3 respectively D.1.4.

1. In the first step, it is analyzed if the query is a SPARQL `ASK` query. This is mandatory – if there is no `ASK` query available, the discovery process is stopped.

2. If an `ASK` query is available, the string is compiled into a SPARQL query and transformed into a SPARQL algebra expression. This is done in order to normalize the queries.

3. Based on the algebra expression, *regular expressions* are deployed in order to identify *concept blocks* from the query (cp. Appendix D.1.3) and extract the minimum DoM for concepts.

4. Two regular expressions are used: The first (cp. Listing D.8) finds the concept blocks, the second (cp. Listing D.9) finds the URI of the semantic concept, the corresponding minimum DoM, and the element, the semantic concept is referenced from (here: input, output).

5. Each concept found is instantiated as a separate object. If a minimum DoM has been specified, it is added to the object, else the minimum DoM for this object is a fail. Objects/concepts are classified into different lists based on the element (here: input, output) the semantic concept is referenced from.

6. Afterwards, minimum DoMs (if any) are deleted from the SPARQL algebra expression. The algebra expression is converted back into a query. With this query, only services, which exactly match the query, can be found.

7. Therefore, the actual matchmaking process is made up from two distinct steps:
   a) In the first step, the revised query is used – as the query does not contain any minimum DoMs, only those services are found, which exactly match the query. Found services are added to the result set and get the similarity value 1.0.
   b) In the second step, services similar to the query are reckoned. The actual approach for this is based on the matchmaking algorithm applied, here: LOG4SWS.KOM, Variant 1c. In the end, a DoM between the query and the services which do not exactly match the revised query has to be calculated. These services are also added to the result set if the defined threshold is met.

8. The result set is sorted based on the similarity values.

9. The result set is handed back to the user. Using the `tModelKeys` from the result set, the user can identify particular services and invoke them.

The need for the two distinct matchmaking steps shown in Step 7 evolves from the fact that SPARQL is a query language for pattern matching. The recognition of similar patterns is not intended in SPARQL. If we assume that we are looking for a particular semantic concept which is referenced from a service input element and we want to find services, whose input elements reference similar semantic concepts, it is necessary to define a query for each similar semantic concept available. If there are four similar concepts (i.e., concepts standing in a subsumption relationship with the “original” concept), it would be necessary to automatically construct five distinct queries. The number of queries would grow disproportionately if more than one concept is regarded. Consequently, this approach is not really applicable.

Hence, the identification of services similar to a query needs to be outsourced from the SPARQL query engine to a matchmaking algorithm. There is no restriction which matching paradigm is applied in the matching algorithm, e.g., it could combine logic and linguistic approaches. However, the matchmaking algorithm needs to be able to accept the semantic concepts from a query stored in separate lists for distinct WSDL elements (cp. Step 5). Furthermore, a numerical similarity value is expected from the matchmaker in order to make the result set sortable.

4.3.5 Implementation Overview

Figure 4.4 shows an overview of the enhanced UDDI designed and implemented in this section. The central component of the implementation is the UDDI registry, which is an enhanced version of *Apache jUDDI*. As it was said before, at the time of implementation, jUDDI has not been compliant with UDDI Version 3.0. Hence, the implementation was conducted for a UDDI Version 2.0-based service registry. However, the differences between these specifications in the context of this thesis are only subordinate

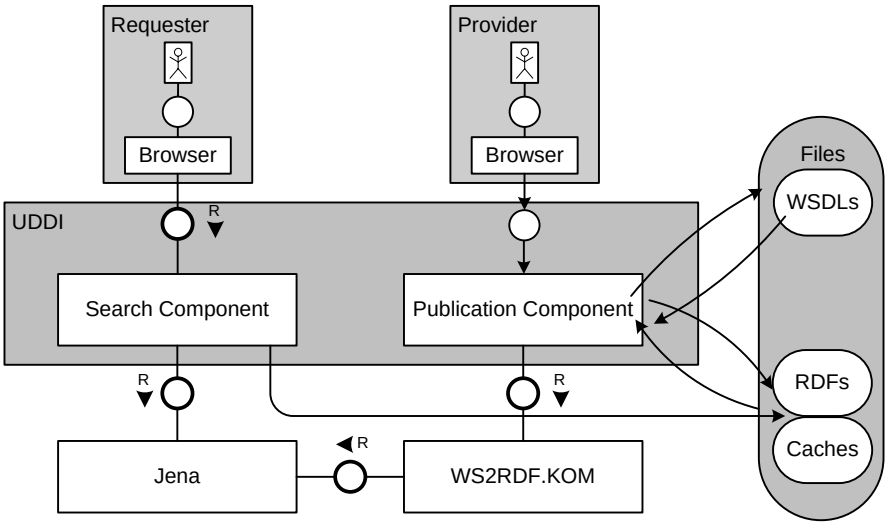


Figure 4.4: Enhanced UDDI – Implementation Overview

(cp. Appendix A.2.1) and it is possible to transfer these results to the newest release of jUDDI or an alternative UDDI implementation.

As it can be seen from Figure 4.4, both service requesters and providers interact with the enhanced UDDI via a Web browser. The search component encapsulates the search process as presented in Section 4.3.4. SPARQL queries are processed using *Jena*, which is also applied in order to transform WSDL-based service descriptions into RDF-based ones. This transformation is handled by the publication component. WSDL and RDF as well as caching files are stored inside a file store. As the RDF and caching files are generated and employed in the same contexts, their storages are linked with each other. For the purposes of Figure 4.4, further software components, e.g., LOG4SWS.KOM or *Pellet*, which are wrapped by the search and publication components, are omitted.

4.3.6 Discussion

If comparing the six requirements defined in Section 4.2 with the work presented in this section, we can make the following statements:

Combination of syntactic and semantic search has been accomplished through the usage of SPARQL and the integration of SAWSDL into UDDI.

Reuse has been accomplished through the usage of SPARQL. However, this solution provides no generic integration approach and is therefore restricted to UDDI.

Ranges have been partly accomplished through the `SIMILARITY`-enhancement of SPARQL. However, this is more a practical than a conceptual approach to define thresholds in a query. Furthermore, ranges/thresholds only apply to semantic similarity DoMs.

Flexible handling of different matchmakers has been partly accomplished: Every matchmaker suitable for SAWSDL might be applied for the non-RDF part of the matchmaking process. However, it is not possible to choose a matchmaker for a certain part of a query or to configure it. Regarding the proof of concept implementation, the matchmaker needs to handle the given DoM thresholds (cp. Section 4.3.4).

Unified query syntax for different service formalisms has not been regarded. Only SAWSDL (based on WSDL 1.1 or 2.0) is a valid service formalism for this solution.

Handling of different registry standards has not been regarded.

Table 4.1: Comparison of Registry Standards ebXML and UDDI (cp. [54, 239])

Category/Feature	ebXML Registry 3.0	UDDI 3.0
Service description standards	WSDL 1.1	WSDL 1.1
Registry	YES	YES
Repository	YES	NO
Object-oriented information model and API	YES	NO
Extensible information model	YES	NO
User-defined queries	YES	NO
SQL query syntax	YES	NO
XML query syntax	YES	YES
JAXR API	YES	YES

As it can be seen, not all the requirements discussed in Section 4.2 have been fulfilled by the work presented here. This can be traced back to the lightweight approach conducted. As we will see in the next section, the usage of different service formalisms makes it necessary to define a common model for services. The same applies to the integration of different matchmakers. In contrast, the solution presented here has shown how it is possible to integrate SPARQL into a UDDI-based service registry using official W3C recommendations. However, this leads to queries which are not clearly arranged – as it can be seen from Listings D.6 and D.7, it is necessary to define a large number of different statements in order to address a single service component, which is not very user-friendly.

Another aspect worth discussing is the applied service registry standard. Even though UDDI has drawn great attention especially from the research community, it suffers from some major drawbacks, making it difficult to use it as a starting point for an advanced query formalism applied in SWS discovery. A comparison of some major features provided by ebXML Registry and UDDI, which are important for the selection of a registry standard, are shown in Table 4.1. For an exhaustive enumeration of the features of both registry standards, the reader is referred to [54, 239].

First of all, it has to be mentioned that both registry standards only support WSDL 1.1-based service descriptions. Hence, it was necessary to introduce a mapping from WSDL 1.1 to 2.0 in the UDDI solution presented above. Second, UDDI provides by default only a registry, where metadata about artifacts are stored. The actual artifacts (e.g., WSDL) are not stored in UDDI. Instead, references to these artifacts are published in the registry. Nevertheless, a repository can be used in addition to UDDI for the management of the artifacts. In contrast to UDDI, the ebXML Registry provides both, a registry and a corresponding repository. Hence, besides the metadata, also the artifacts themselves are published in ebXML. Third, UDDI makes use of a relatively flat data model, which cannot be extended, whereas ebXML Registry offers an object-oriented and extensible information model and API. Fourth, both registries can be used by utilizing the Java API for XML Registries (JAXR), which provides a uniform way for communicating with a registry. Finally, concerning the search facilities of both registries, ebXML has enhanced querying capabilities by providing SQL support and user-defined queries in comparison to UDDI, which is only able to process XML-based queries. Regarding the integration of an advanced query formalism into a service registry, this is a major point. All things considered, UDDI only offers a subset of the features provided by ebXML Registry [54, 239], and therefore, ebXML has been chosen for the approach presented in the next section.

4.4 A Unified Querying Formalism for Semantic Web Services

Following the findings from Sections 4.2 and 4.3.6, an advanced approach to query formulation based on an existing language has been designed, developed, and implemented for ebXML Registry. This section presents the conceptual design of the querying approach (Section 4.4.1), the enhancement of SPARQL in SWS2QL based on this approach (Section 4.4.2), the conceptual integration into ebXML Registry (Section 4.4.3), and the implementation of the overall framework (Section 4.4.4).

4.4.1 Design of a Unified Querying Formalism for Semantic Web Services

In addition to the requirements addressed in Section 4.2, which focus on the capabilities and integration of a querying approach for SWS, the query language itself has to fulfill different criteria [1, 38]:

Expressive power: Concerning the expressiveness, a subset of the requirements, which have been often used in the context of RDF query languages, can also be adapted to other query languages [38]. This subset comprises:

- An appropriate, but also powerful expression syntax for navigating the component hierarchies, which are contained in the data.
- Functionality for querying the components and their attributes.
- Support for data types and a comparison of values.
- Capability to deal with optional parameters.

Schema awareness: A query language should be schema-aware, i.e., aware of the data model, a query is referred to. When queries are defined, this allows for the exploitation of the underlying schema for different purposes, e.g., for type checking or optimization. In addition, this structure consciousness is closely tight with the awareness of the underlying semantics and the requirement for expressive power.

Semantics: Direct consequences of imprecise semantics are ambiguity and misunderstandings. Therefore, a formal description of the semantics of the query formalism is required.

Program manipulation: In order to allow for a manipulation (e.g., translation) of queries, the query language has to be machine-processable, i.e., the query language, with respect to syntax and semantics, must be simple enough and unambiguous to permit an automated processing of a query. However, a trade-off exists between user-friendliness and simple parsing, which also has to be considered. In doing so, a simple and convenient structure has to be established while avoiding redundancy and sustaining readability.

Compositionality: This requirement addresses the decomposition potential of queries, so that larger queries can be split into a number of smaller queries. For this, the query language must permit to reuse the output of one query as the input of another query.

These requirements show that it is necessary to address two different aspects in order to enable a query formalism for SWS: First, the requirement for *schema awareness* shows the necessity to provide a data model the queries will be based on. Second, the query language itself has to be modeled in terms of its syntax, *semantics*, and *expressive power* in general. *Compositionality* is not explicitly regarded in the solution at hand, but could be addressed in further versions in order to optimize queries.

The solution at hand (cp. Section 4.4.2) exploits SPARQL as foundation for a sophisticated SWS query formalism. Hence, a number of the abovementioned criteria is to some extent adopted from SPARQL, especially the *expressive power*. Nevertheless, the expressive power of SWS2QL is different from SPARQL, as both address different data models – accordingly, we will highlight the corresponding characteristics of SWS2QL.

In the following, we will present the both models which provide the foundation of SWS2QL: The data model and the abstract query model.

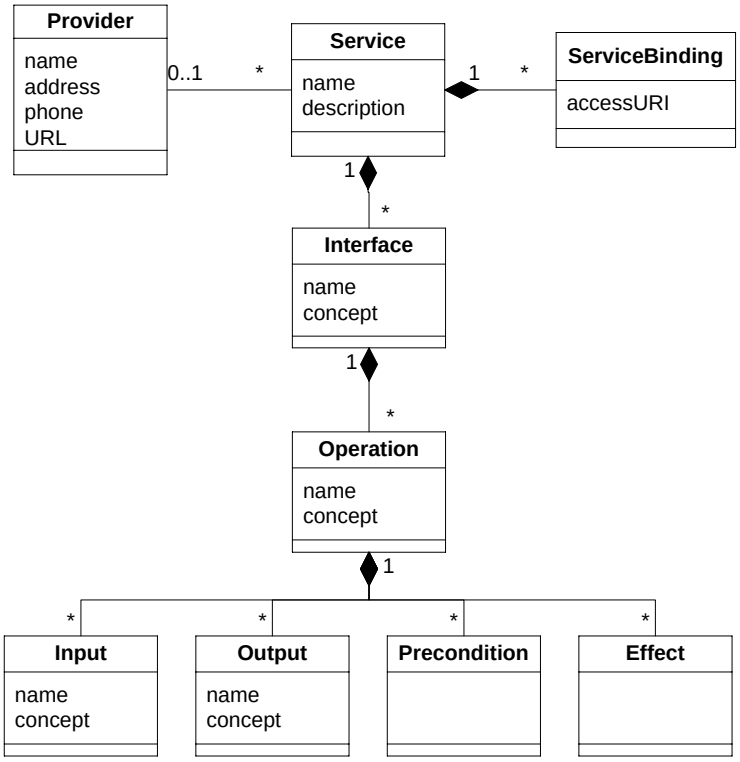


Figure 4.5: Abstract Data Model for Services

A data model for SWS queries describes the components of a service as well as their properties and relationships to other service components. In Chapter 3, ATSM has been introduced, which is a common model for SAWSDL and OWL-S (cp. Appendix A.1.3). Hence, it already comprises the most important service components applied in service discovery but is restricted to those components usually needed in matchmaking. For its usage as foundation for query formulation, it needs to be slightly enhanced:

First of all, service requests are often stated by specifying a description of the "perfect" service example (cp. Section 4.1). In general, the creation of such a query by example will be probably restricted to more sophisticated service requesters. Nevertheless, the opportunity to specify an already known service in order to search for similar services (e.g., as a substitute for a given service) has to be considered within the query model. Therefore, the `ServiceBinding` component is also adopted into the model in order to be able to specify the access URI of the respective service. Furthermore, a service requester may also be interested in services offered by a specific service provider. This information is also provided by common registry standards. Therefore, the data model also incorporates a `Provider` component, even though it is not part of the ATSM service description. Furthermore, the structure of input and output parameters in ATSM and the data model at hand differs slightly. While in ATSM we differentiate between inputs/outputs (respectively message references, cp. Appendix A.1.3), for SWS2QL, input and output parameters are direct subelements of an operation. As the research community is still debating about a common format for preconditions and effects in SWS formalisms like, e.g., SAWSDL and OWL-S, we consider these elements in our abstract data model for services. However, we do not define any attributes for these service components.

Figure 4.5 shows an overview of the resulting service model. In general, the data model makes no claim to be complete, but aims at providing a self-contained, lean data model or schema capturing the most relevant information of a service, which can be individually customized. This data model is applied as an example and could be replaced by another model; of course, this would make it necessary to change the applied query syntax (see below).

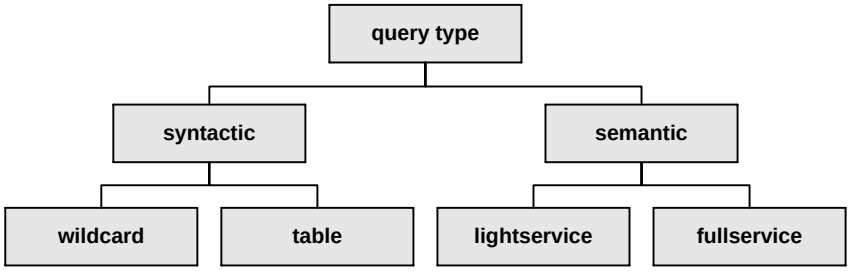


Figure 4.6: Overview of Query Types

Based on the data model provided, it is possible to define an abstract model for queries, which provides the foundation to exploit and enhance an existing query syntax. In general, queries based on common query languages (e.g., SQL or SPARQL) encompass different sections (or *components*), which can be summarized to a *result parameter* component, a *query statement* component, and a component to specify *solution modifiers* for altering the respective result set [39]. Regarding SPARQL, these different components would be a *pattern matching part*, *solution modifiers*, and *output*. The latter is the specification of the variables of interest [39]. Since the first and the last component both refer to the results of a query, the enumeration can be shortened to a query statement component and a result component. The structure of both parts is discussed in the following.

Before elaborating on the semantics of the query statement component, it is necessary to address common use cases with respect to query types in order to derive a generic query statement structure. For this, four basic query types are used, which are derived from the discussion of common approaches to query formulation in Sections 2.3.1 and 4.1. The query types are depicted in Figure 4.6. The first two query types are subtypes of syntactic, keyword-based approaches, while the last two types represent semantic-based approaches. In detail, *wildcard* search allows the user to define single terms as well as placeholders, while *table*-based search additionally permits to state attribute-value pairs [127]. The *fullservice* approach is used to refer to a complete service description representing the service request, i.e. a "query by example", while the *lightservice* approach allows for the specification of arbitrary excerpts of a complete service description according to the needs of a service requester. For example, a fullservice query can be established by specifying the access URI of a service, so that the corresponding service description can be retrieved. In contrast, a lightservice query can be used to specify the properties of arbitrary elements like single interfaces or single operations combined with the input and output parameters a service offer has to provide.

As already mentioned in Section 4.2, a flexible query mechanism should also address the dynamic selection of different matchmakers controlled by the service requester, so that appropriate matchmakers can be applied for specific matching purposes. In doing so, the support for a comparison of values is enhanced or rather generalized, which increases the expressive power of the query language (see above). Within the work at hand, the four query types are used to exemplarily refer to different matchmakers. For example, in order to process the information based on one of the keyword-based approaches, the default registry capabilities could be applied, while the semantic-based approaches could be directed to a custom semantic matchmaker. Of course, it is also possible to integrate other matchmakers into the service registry and therefore reference in a query.

In order to be able to associate the respective statements of a query to their corresponding matchmakers, *query sections* are introduced representing an instance of a specific query type. Besides the definition of multiple query sections referring to different query types within a service request, also the combination (e.g., conjunction, inclusive or exclusive disjunction) of these sections needs to be addressed within a query. For this, query sections have to be organized in some structure, i.e., a *global query container*, indicating the type of combination. The capabilities of query sections address the requirement of compositionality (see above) of a query language, since single query sections can be specified or several query sections can be combined to a complex query construct.

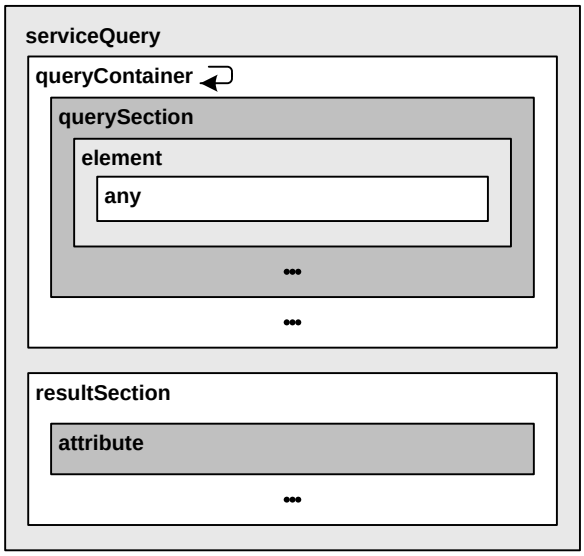


Figure 4.7: Abstract Query Model

In general, the actual query parameters are specified within a query in terms of query statements using the respective query syntax and commonly refer to specific attributes of a component of the underlying data model, which also addresses the expressive power of a query language. In general, the values of properties can be expressed using arbitrary data types. Nevertheless, a specific data type needs to be defined for each property in order to establish a machine-processable request.

In addition to the criteria of the request, the result parameters and the properties of the result set need to be specified within a query. The result parameters specify the desired objects to be returned and the properties may refer to different aspects of the result set (e.g., the maximum number of results, sorting criteria). Both parts can be defined within the *result section*. Based on the components of a unified service query, an abstract query model is introduced, which serves as a blueprint for service queries: Its overall structure has been defined as depicted in Figure 4.7. Basically, service queries based on the model comprise a global query container and a result section. The global query container may either contain further query containers or query sections. In the latter case, the query sections are interconnected using a single combination type (`AND`, `OR`, `XOR`) defined in the global query container, whereas in the first case, a nested structure of query containers is established in order to organize the query sections using arbitrary combination types.

Within a query section, an arbitrary number of elements of any type can be stated representing the arguments of a query. Regarding the combination of the query elements within a single query section, a conjunction (i.e., `AND`) is assumed for two reasons: On the one hand, a query section can be considered as a semantic unit, where all elements belong to the same parent element (e.g., all input and output parameters defined within a query section belong to the same operation), and on the other hand, the introduction of additional combination types within a single query section would increase complexity and is therefore not very user-friendly.

In general, thresholds are defined in order to narrow the potential result set (cp. Section 4.2). This can be achieved by a service requester by prescribing a minimum level of similarity with respect to the properties or even the whole service offer. For this, the opportunity to define similarity thresholds is introduced for both, i.e., each element within a query section, as well as for the whole service. The actual determination of similarities is accomplished by the responsible matchmaker and thus, the data type of the threshold value depends on the selected matchmaker. Therefore, threshold values are not restricted to numerical values but can also be expressed in terms of text-based constants. This allows for the utilization of similarity functions from the field of IR to determine the syntax-based similarity of terms as well as the application of reasoning techniques by specifying DoM values [125, 211].

Concerning the result section, an arbitrary number of attributes can be specified, which refer to the desired result parameters to be returned. In addition, properties of the result set can be defined within

this section (e.g., a descending or ascending order of the results with respect to some attribute of the result set).

Besides the result set modifiers, the attributes which should be contained within the result set are also defined (e.g., the name of the service or the provider name). One of these attributes can also be used as sorting criterion by specifying its ID as the value of an `orderBy` attribute. The abstract query model is defined using an XSD document, which is presented in Appendix D.2.

4.4.2 From an Abstract Query Model to SWS2QL

Within this section, the SWS Structured Query Language (SWS2QL) is presented. Basically, SWS2QL represents an enhancement of a common query syntax, namely SPARQL, on the basis of the data and query model presented above. Apart from the reasons to apply SPARQL as query language in SWS discovery already presented in Section 4.3.1, it also provides a much more user-friendly approach to query formulation in comparison to the ebXML Registry SQL-query syntax as well as the SPARQL queries based on the official W3C mapping from WSDL to RDF (cp. Section 4.3 and Appendix D.1). By reusing a common query syntax, the requirement for program manipulation of query languages has been met (cp. Section 4.4.1), but potential enhancements of the query language must also conform to the SPARQL grammar to maintain program manipulation. Concerning the enhancements of SPARQL, the following requirements are addressed in addition to the requirements for an SWS query language already outlined in Sections 4.2 and 4.4.1:

- Compatibility with the existing SPARQL grammar.
- Reuse of existing extension principles.
- Minimization of the amount and complexity of further rules.

First of all, in order to specify SWS2QL queries, the `SELECT` form of SPARQL queries is utilized, since data of a Web service description should be returned to a service requester, instead of a solution graph or a boolean value. According to the query structure presented in Figure 4.7, an SWS2QL query must have means to define `querySection` elements and a `resultSection`. The latter is accomplished by specifying variables after the `SELECT` keyword to define the result parameters and by using the solution modifiers provided by SPARQL to set constraints on the result set. Regarding query sections and their assignment to different matchmakers in accordance with the abstract query model, a special graph pattern of SPARQL, the *RDF Dataset Graph Pattern*, is utilized, which per se facilitates the grouping of triple patterns to be matched against a specific, named RDF graph (cp. [215] and Appendix A.3). This normally allows for the matching of different parts of a query against different graphs [118]. Since RDF-based service descriptions are not considered within the approach at hand, the specification of a named RDF graph using the RDF Dataset Graph Pattern is utilized to assign the associated query section to a specific matchmaker. Thus, multiple matchmakers can be stated within a query, each processing its assigned part of the service request. Here, the concept of default and named graphs is applied [215] (cp. Appendix A.4). For this, four specific predicates in accordance to the abovementioned query types, i.e., `fullservice`, `lightservice`, `wildcard`, and `table`, are introduced for named graphs. The value of a new predicate denotes a specific matchmaker of the query type represented by the predicate. In general, the RDF Dataset Graph Pattern is applied by specifying the `GRAPH` keyword of the SPARQL grammar, which can either refer to an IRI of a graph or to a variable ranging over the IRIs of all named graphs of the query's RDF dataset. Outside the query section embraced by the `GRAPH` keyword, RDF triple patterns are matched against the default graph [118, 215].

Although, SPARQL and the presented abstract query model allow for an arbitrary combination of graph patterns or query sections, SWS2QL is restricted in the following to at most one conjunction of two query sections for clarity. Furthermore, it is assumed that in case of a conjunction, a syntactic query type is combined with a semantic query type, which is most likely to be applied in practice. However, depending on the matchmakers supported by a registry, SWS2QL can also be enhanced with further query and combination types. Admittedly, a service requester has to be aware of the supported query types. In contrast, the supported matchmakers do not have to be necessarily known by a service requester. In this case, a default value can be specified for the query type predicates indicating the application of the default matchmaker.

Listing 4.1: Example SWS2QL Query (Excerpt)

```
1  SELECT ?service
2  WHERE {
3     ?table aqm:table DEFAULT .
4     GRAPH ?table {
5        ?service rdf:type atsm:service .
6        ?service atsm:hasname ''weather forecast'' .
7     } ...
```

Within the query sections of SWS2QL queries, the properties of the components of the desired service are specified in terms of RDF triples. An excerpt of a simple, table-based SWS2QL query is shown in Listing 4.1. Prefix definitions are skipped for ease of reading. In order to permit imprecise matching and to integrate similarity thresholds into an SWS2QL query, a specific extension function is introduced, which can be used within SPARQL `FILTER` expressions. Generally, the opportunity to make use of extension functions is inherently provided by the SPARQL grammar and has also been proposed by Kiefer et al. [125] for the integration of imprecise matching capabilities into SPARQL (cp. Section 4.5).

Listing 4.2: Extension Function for Determining DoM Values in SWS2QL

```
xsd:boolean sws2ql:dom(IRI offURL, IRI reqURL, simple literal dom)
```

In SWS2QL, the extension function for the integration of semantic references is defined as shown in Listing 4.2. Basically, this function takes into account two URLs and a DoM value, which can either be text-based or numerical. Depending on the type of matchmaker associated with the surrounding query section of the extension function, the semantics of the URLs change. In case of a lightservice query, the URLs represent semantic references pointing to OWL DL-based concepts of a specific component within the service offer and the service request. Both are to be compared with the specified DoM value in mind. If the extension function is specified within a fullservice query section, the URLs refer to the access URIs of the service offer and the query by example service and are used by a fullservice matchmaker to identify two services. In general, the DoM value can either be one of the predefined DoM constants or a numerical representation of the requested degree of similarity.

In detail, the first argument of the extension function is given by the respective variable of an RDF triple, which is matched against all service offers, and the second argument is specified by the service requester in order to state the desired OWL DL-based concept for a specific property or to reference the example service to retrieve similar services.

Based on the different threshold values, a matchmaker is able to determine the matching services for a specific query section. Although, weights could be introduced for the several parts of a service within a single query section, this would raise the modeling effort and complexity. Thus, the computation of a total similarity value for the services resulting from a single query section is left to the respective matchmaker within SWS2QL. However, a service requester must have means to obtain the total similarity value of a specific query section in order to specify global similarity thresholds. As already stated by Kiefer et al. [125], the application of extension functions does not permit to assign similarity scores to variables, so that similarity values cannot be reused within other query statements, e.g., for the weighted aggregation of the results. However, assigning similarity values to variables can be achieved within SWS2QL through the application of the `GRAPH` keyword.

Listing 4.3: Property Function to Retrieve Total Similarity Scores in SWS2QL

```
Var1 sws2ql:sim (IRI graph)
```

In order to obtain the total similarity value from a matchmaker, the virtual triple approach as proposed by Kiefer et al. [125] is used (cp. Section 4.5). In doing so, the property function depicted in Listing 4.3 is introduced. This function takes into account one argument, the object value of the RDF triple, representing the IRI of the graph assigned to a specific query section. The global similarity value computed by the matchmaker associated with the query section is then returned by the function and assigned to

the subject variable. For the weighted aggregation of the results of all query sections, a similar property function to the one proposed by Kiefer et al. could be used. However, as already mentioned, SWS2QL is restricted to at most one conjunction of a semantic-based query type and a syntactic query type.

Listing 4.4: Extension Function for the Definition of Global Thresholds in SWS2QL

```
xsd:boolean sws2ql:thold(Var1 sim, simple literal thold)
```

In addition, it is assumed that no fuzzy keyword-based search is performed, i.e., no text-based similarity measures are applied. Nevertheless, a standard `FILTER` function provided by SPARQL can be applied to perform a substring matching. In doing so, the corresponding parameter value of a service offer only has to contain the specified keyword, i.e., the keyword can be part of a larger string, but the parameter value does not have to match the keyword exactly. Since no fuzzy keyword-search is contained within the query, no aggregation function has to be applied for the results of the semantic-based query section and the keyword-based query section. Instead, a service offer must contain the parameters defined in the keyword-based query section and, at the same time, adhere to the arguments and their given threshold values specified within the semantic-based query section. Nevertheless, to be able to specify a global similarity threshold for the total similarity value resulting from a semantic-based query section, a service requester can make use of the new extension function depicted in Listing 4.4. The function takes into account a variable holding the total similarity score of a specific query section and a numerical or text-based value specifying the requested minimum degree of similarity. All services with a total similarity at least as high as the similarity threshold are returned to the service requester.

Listing 4.5: SWS2QL Query (Example)

```
SELECT ?service                                                                    1
WHERE {                                                                            2
   ?lightservice aqm:lightservice MM2 .                                            3
   GRAPH ?lightservice {                                                           4
      ?service rdf:type atsm:service .                                             5
      ?service atsm:hasInterface ?interface .                                      6
      ?interface atsm:hasOperation ?operation .                                    7
      ?operation atsm:hasInput ?input .                                            8
      ?operation atsm:hasOutput ?output .                                          9
      ?input atsm:hasParameter ?inconcept .                                        10
      ?output atsm:hasParameter ?outconcept .                                      11
      FILTER(sws2ql:dom(?inconcept,                                                12
       ''http://www.example.com/geography.owl#location'', ''SUB''))                13
      FILTER(sws2ql:dom(?outconcept,                                               14
       ''http://www.example.com/weather.owl#temperature'', ''EXACT''))             15
   } .                                                                             16
   ?table aqm:table DEFAULT .                                                      17
   GRAPH ?table {                                                                  18
      ?service rdf:type atsm:service .                                             19
      ?service atsm:hasInterface ?interface .                                      20
      ?interface atsm:hasOperation ?operation .                                    21
      ?operation atsm:hasName ?opname .                                            22
      FILTER regex(?opname, ''forecast'')                                          23
   } .                                                                             24
   ?similarity sws2ql:sim (?lightservice)                                          25
   FILTER(sws2ql:thold(?similarity, ''0.5''))                                      26
}                                                                                  27
ORDER BY DESC ?similarity                                                          28
LIMIT 20                                                                           29
```

Listing 4.5 shows an example SWS2QL query. This example combines a lightservice query (Lines 3–16) with a table-based query (Lines 17–24). Regarding the former, Line 3 defines the named graph as described above. In this example, a named graph "lightservice" is defined and the matchmaker *MM2* is applied to it. The query statement itself (Lines 5–11) follows the ATSM specification as presented in Appendix A.1.3. In Lines 12–15, two filters are applied which define the parameter constraints as presented in Listing 4.2. In this example, the service offer's input parameter needs to reference "`http://www.example.com/geography.owl#location`" or be one of its subconcepts (indicated by the DoM "`SUB`"). Accordingly, the service offer's output parameter needs to reference "`http://www.example.com/weather.owl#temperature`", as the corresponding DoM is "`EXACT`". In this exam-

ple, we make use of the subsumption relationships from Section 3.2.1 and the matchmaker `MM2`. However, the filter could also specify a numerical value – in this case, another matchmaker would have to be applied.

The table-based query statement is also defined by specifying an accordingly named graph (Line 17). Here, the filter (Line 23) makes use of the standard *regex* regular expression which defines that the given argument (here: "forecast") needs to be contained in the parameter's textual description (here: name of the operation) [215].

The property function in Line 25 is used to retain the total similarity value for a service computed by the respective matchmaker. For this, the property function defined in Listing 4.3 is used. In order to specify a global similarity threshold value for the service offers, the extension function defined in Listing 4.4 is applied (Line 26). So, in the example at hand, the overall similarity needs to be larger than "0.5". Accordingly, the matchmaker MM2 needs to be able to handle subsumption relationships for single parameters and compute a numerical similarity. In the work at hand, this is done using LOG4SWS.KOM. Finally, solution modifiers are defined in Lines 28 and 29 to display the results in descending order and to restrict the number of results to 20.

4.4.3 Integrating SWS2QL into ebXML

As the approach presented in this section is more advanced than the one presented in Section 4.3, the integration of the new query formalism is more complex. SWS2QL comprises several components, which have to be integrated into the registry standard (here: ebXML Registry; cp. Section 4.3.6). Although the integration approach is per se independent from a particular ebXML Registry implementation, the chosen framework *freebXML* influences some parts of the integration approach. If this is the case, it will be explicitly stated in the following. The components are:

SWS descriptions need to be integrated into ebXML Registry in order to make them available for searching and invocation. Although the querying approach presented above does not depend on a specific SWS formalism, SAWSDL has been used in the proof of concept implementation. The integration of other SWS formalisms can be conducted in a similar way. In accordance with the querying approach, a SAWSDL/WSDL-based service description is published in two different ways: The standard publication mechanism is applied for the native WSDL parts of the service description while an enhanced publication mechanism is used in order to map SAWSDL to ATSM, which is the desired formalism for matchmaking and querying with SWS2QL. The mapping to ATSM is hidden from the user.

SWS2QL is integrated into ebXML Registry by making use of the already available query format, i.e., the SQL search interface. A mapping between SWS2QL and SQL is conducted on client-side, while the actual processing of the queries is performed on registry-side.

Matchmaking facilities need to be integrated in order to make use of distinct matchmakers from within SWS2QL queries. Since ATSM, which is the SWS formalism of LOG4SWS.KOM and COV4SWS.KOM, is integrated into ebXML Registry, it is possible to reuse the matchmakers presented in Chapter 3. However, the matchmakers still need to be integrated into the service registry. Here, LOG4SWS.KOM is applied as proof of concept but could be easily exchanged by other matchmakers, which are able to handle ATSM. This is facilitated by providing a standard interface for the integration of matchmakers into ebXML Registry (cp. Appendix D.2.3).

This way, the registry is able to provide semantic matching capabilities, which makes it necessary to address the handling of ontologies. Newly published service descriptions could reference semantic concepts from previously unknown ontologies, requiring the integration of new ontologies into the registry. As it was mentioned in Chapter 3, reasoning over semantic concepts and their classification are very expensive processes, which was therefore addressed by incorporating caches. In this approach, we go one step further and integrate a semantic reasoning engine in conjunction with an ontology knowledge base into the service registry. If a query is enhanced with semantic information, the syntax-based part can be directed to the standard search facilities provided by a registry and the additional semantic information can be directed to semantic matchmakers to allow for reasoning support. In the following, the three different integration steps will be presented.

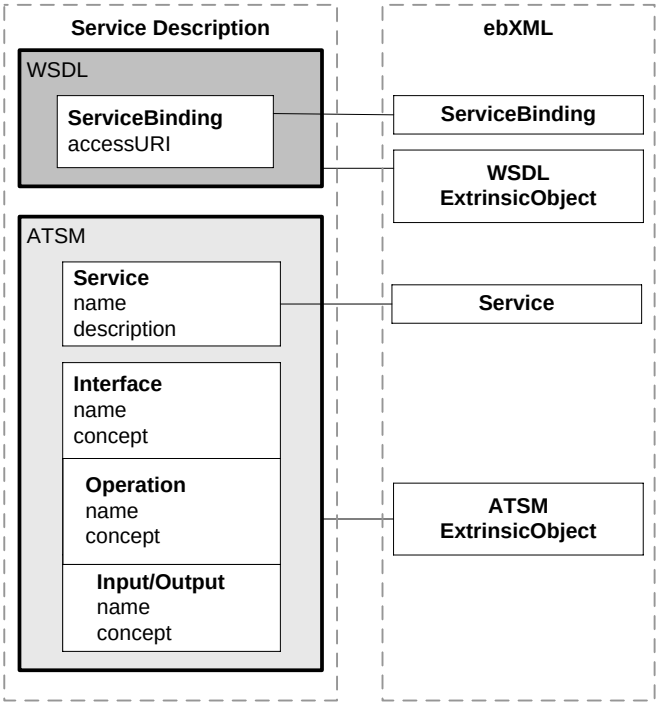


Figure 4.8: Mapping SWS Information to the ebXML RIM

In order to be able to retrieve SWS descriptions, they have to be published in the registry. Since common registry standards are not able to cope with SWS descriptions by default, an appropriate concept for their integration is required. In the solution at hand, this integration is based on ATSM:

It is assumed that SWS description formats are transformed into ATSM documents and that the corresponding WSDL information of the respective SWS description format is published in the usual way, i.e., the standard publication mechanism provided by the respective registry is utilized. The retrieval of an associated WSDL file can be achieved automatically, since the WSDL file information is represented (in SAWSDL) by an SWS description. In order to store the ATSM documents within a registry, existing approaches for the integration of external description documents into registries can be applied. In case of a UDDI registry, external description documents can be referenced using tModels (cp. Section 4.3.3). Concerning ebXML Registry, the ATSM documents can be stored as extrinsic objects, i.e., the same way WSDL documents are stored (cp. Figure 4.8 and Appendix A.2.2). But in contrast to a WSDL file, no mapping of ATSM information to further ebXML Registry Information Model (RIM) classes is necessary. In doing so, SWS descriptions are transformed to ATSM representations which are stored in the registry, and the corresponding WSDL information is published in the usual way.

Before elaborating on the details of the publication process, the necessary changes and preparations with respect to the registry implementation and configuration are addressed. Since a registry must have means to distinguish SAWSDL-based documents, which are used for the proof of concept implementation for the description of service offers, from standard WSDL-based service descriptions, a new classification node “SAWSDL” has to be created in the registry. In addition, an “ATSM” classification node has to be created for the classification of the ATSM representations. Depending on the type of content (e.g., WSDL), an associated cataloging service, which extracts the required information from submitted documents, is normally invoked on publication time. The information is then mapped to instances of the ebXML RIM representing the content’s metadata, while the actual content is stored within the repository (cp. Appendix A.2.2). Instead of the documents themselves, references to external content descriptions can be submitted to the registry, too. In this case, the content is not stored within the repository. The registry then only stores the metadata and a link to the external content.

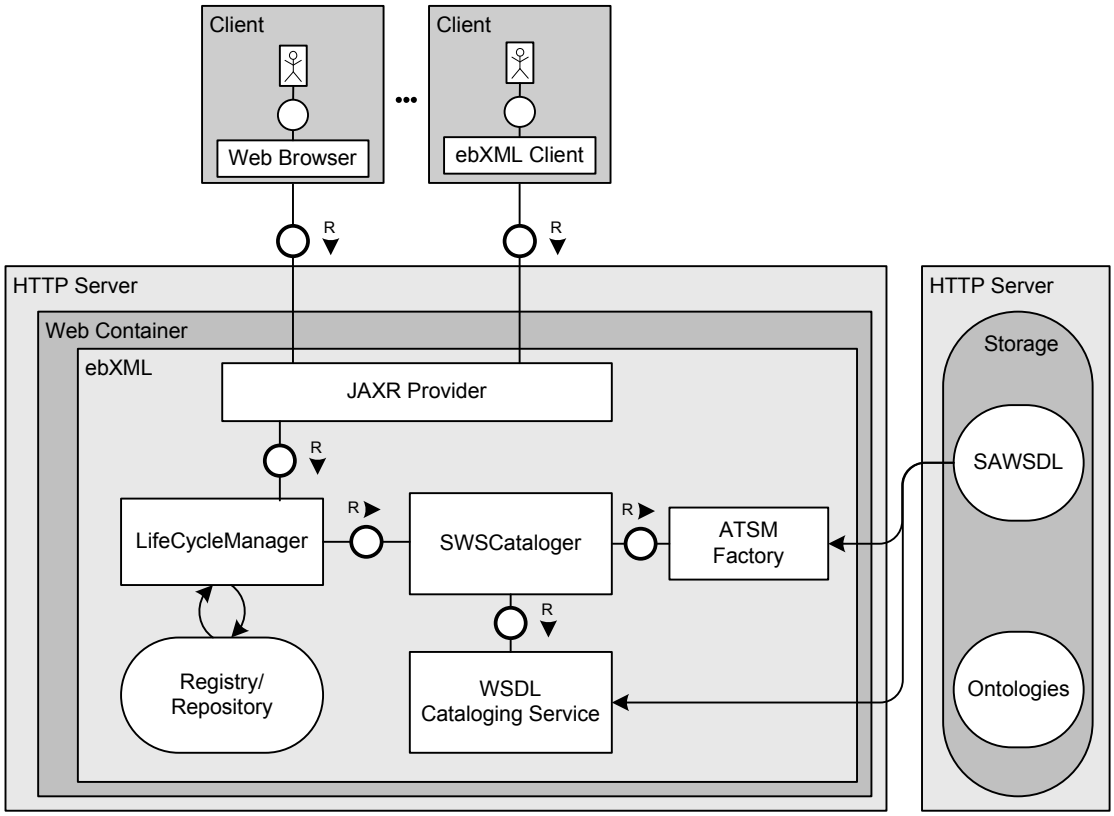


Figure 4.9: Publication in Enhanced ebXML-based Service Registry

A detailed description of the publication process of SAWSDL-based documents in freebXML is presented in the following. The major steps of the process are depicted in Figure 4.9. Basically, the ebXML Registry standard provides support for WSDL 1.1 service descriptions. Hence, SAWSDL (WSDL 1.1) documents which are stored on an external Web server, are submitted to the registry as "ExternalLink" objects and classified with the new object type "SAWSDL". Subsequently, the *LifeCycleManager* of the registry is invoked, which calls the appropriate cataloging service associated with the "SAWSDL" object type. For this purpose, a new cataloging service, the *SWSCataloger*, has been developed. On invocation, the SWSCataloger first makes a call to the WSDL 1.1 cataloging service of freebXML, which performs a normal publication of the WSDL information contained in the SAWSDL document. Afterwards, the SAWSDL document is retrieved from the external server and transformed into its ATSM representation. Since the utilized ATSM parser implementation is only able to cope with SAWSDL documents, which are based on WSDL 2.0, the SAWSDL documents are transformed into (SA)WSDL 2.0 documents using the XSLT stylesheet presented in Appendix A.1.2. Since the resulting ATSM service object cannot be directly stored, it is serialized into a temporary XML file using the Java Architecture for XML Binding (JAXB) 2.0[2]. This file can then be stored within the registry as an extrinsic object. Finally, the published WSDL information has to be associated with the ATSM representation of the SWS. For this, ebXML provides the ability to relate two objects using arbitrary relationship types. Within the prototypical implementation, a "contains" relationship is used to associate service RIM objects with their corresponding ATSM representation. The resulting registry and repository objects created by both cataloging services are then passed to the *LifeCycleManager*, which submits the content to the relational database of the service registry.

Integration of SWS2QL

Possible integration approaches for SWS2QL into freebXML are restricted by JAXR, which serves, amongst other functionalities, as query interface for freebXML. However, both JAXR and the registry support SQL.

[2] `http://jaxb.dev.java.net/`

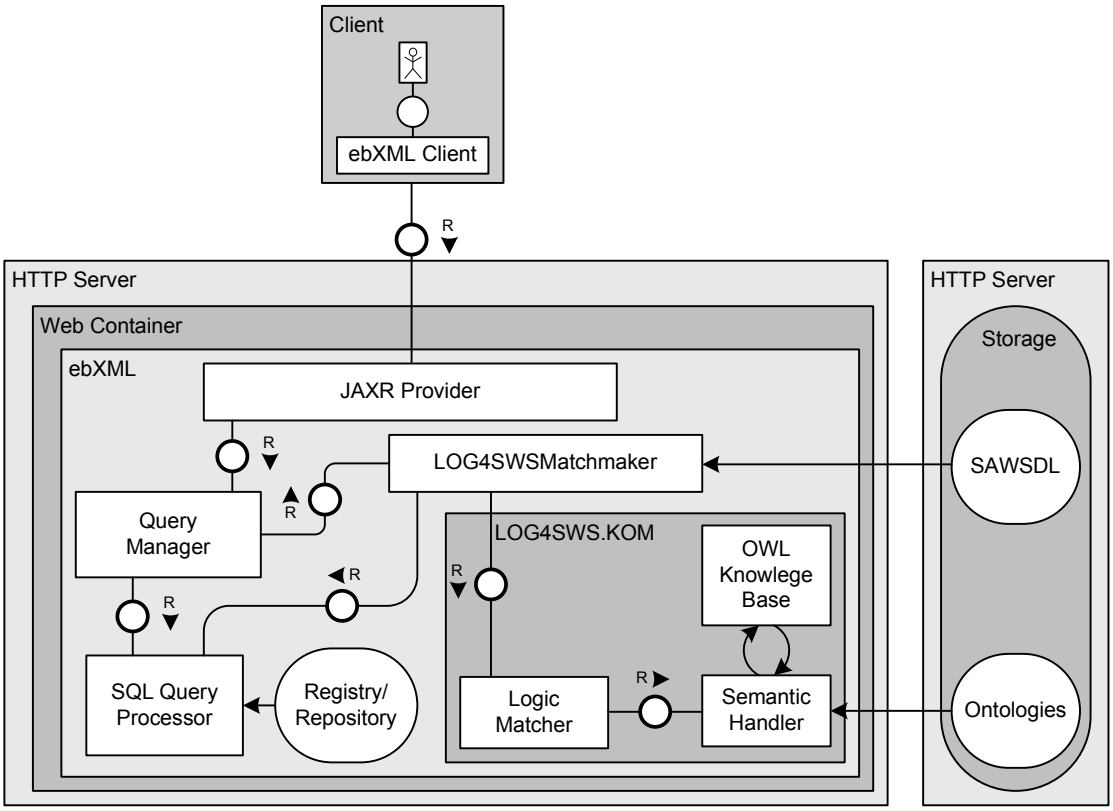


Figure 4.10: Discovery in Enhanced ebXML-based Service Registry

Here, native SQL is intended to be executed against the table of the relational database system, which provides the data management in the registry. Hence, we enhance the usually applied SQL queries by providing a native and an enhanced part. The latter refers to the semantic information defined in an SWS2QL query. The keyword-based query sections of SWS2QL are mapped to the native part, while the enhanced part is used to represent the service description-based parts of SWS2QL. A transformation of SWS2QL queries to enhanced SQL queries is accomplished on client-side, while the actual processing of the queries is performed on registry-side. The structure of the SQL enhancement as well as the structure of the SQL query statements are presented in Appendix D.2.

Concerning the mapping of a service description-based query section of an SWS2QL query to the corresponding SQL query statement as part of the SQL enhancement, each `FILTER` statement within the SWS2QL query is represented by a parameter within the SQL query statement. Furthermore, each parameter itself comprises the required attributes of the respective `FILTER` statement and if necessary, the service component level, the parameter is referred to. In addition, a final parameter is added to the SQL query statement specifying the desired matchmaker (e.g., LOG4SWS.KOM). The actual processing of the enhanced SQL queries is performed on registry-side and presented in the next section.

Integration of a Matchmaker

The retrieval process in the enhanced service registry is depicted in Figure 4.10. First of all, an SWS2QL query is created by a service requester. On client-side, the SWS2QL query is transformed into an enhanced SQL query, which is submitted to the *Query Manager* of the registry using JAXR. Afterwards, the native part of the SQL query is processed by the standard *SQL Query Processor* provided by the registry. Subsequently, the result set of the native SQL query and the extension of the native SQL query string specifying the semantic parts of the query are passed to the desired matchmaker stated within the request. For the prototypical implementation, LOG4SWS.KOM (Variant 2c) has been applied. Details of the required matchmaker interface are presented in Appendix D.2.3.

The matchmaker queries the registry using the SQL query processor in order to retrieve the required ATSM descriptions for each service within the result set of the native SQL query. In case of a fullservice query, a matchmaker must eventually retrieve the description of the example service from an external server, if the description is not already contained in the registry. Afterwards, the matchmaker processes the ATSM service descriptions according to its specific matchmaking algorithm and the query parameters contained within the SQL extension. Subsequently, the matchmaker may alter the primary result set and may return additional information for each service object in the final result set using a response slot list provided by freebXML, which is part of the result object. Finally, the result is sent back to the ebXML client.

4.4.4 Implementation Overview

For the actual implementation, *freebXML* 3.1 was used and enhanced. freebXML is an open source reference implementation of the OASIS ebXML registry standards [85, 86]. freebXML is made up from a registry, where metadata about artifacts can be published, and a repository, where the actual artifacts are stored. In general, an ebXML registry may implement different profiles, i.e., provide functional enhancements for a specific type of content. Concerning profiles, freebXML implements, among others, the *ebXML Registry profile for Web Services* [191], which allows for the publication, management and retrieval of WSDL 1.1-based service descriptions. A comprehensive presentation of the freebXML registry architecture is given at the project's Web page[3].

The following major components have been altered or added in order to implement the integration as presented in the last section: First of all, a new cataloging service has been developed for the publication of SAWSDL-based documents within freebXML. For the integration of SWS2QL, a separate freebXML client has been developed and the query manager implementation of freebXML has been modified. For the actual matchmaking process, it was necessary to integrate a reasoning engine into the registry. In LOG4SWS.KOM, which is the applied matchmaker, a *Semantic Handler* provides an interface to the reasoner as well as to the OWL knowledge base (cp. Figure 3.5). An instance of this handler has been integrated into the *RepositoryManagerFactory*, which represents a permanent and unique instance within the freebXML registry, so that the handler and the corresponding OWL knowledge base can be initialized once upon registry startup.

4.4.5 Discussion

If comparing the six requirements defined in Section 4.2 with the work presented in this section, we can make the following statements:

Combination of syntactic and semantic search has been accomplished through the usage of SWS2QL and its integration into ebXML, and the applicability and integration of different matchmakers.

Reuse has been accomplished through the usage of SPARQL-based SWS2QL, which can be easily transferred to other service frameworks.

Ranges have been accomplished through the definition of thresholds in SWS2QL.

Flexible handling of different matchmakers has been accomplished: Every matchmaker implementing the interface presented in Appendix D.2.3 might be applied.

Unified query syntax has been regarded through the application of an abstract service data model and an abstract query model.

Handling of different registry standards has been regarded: Even though the proof of concept implementation has been conducted for ebXML registries, the abstract models as well as SWS2QL might be integrated in UDDI or proprietary registries, too.

[3] http://ebxmlrr.sourceforge.net/

In summary, SWS2QL represents a unified query language for the retrieval of SWS and addresses the shortcomings of other querying approaches. First of all, SWS2QL provides an advanced and at the same time very flexible querying approach, since it is based on a lightweight content model and generic query structure, which can be easily customized to the individual needs of a service requester. In contrast to other approaches, SWS2QL does not force service requesters to express their service requests by specifying a query by example service description (e.g., [170, 260]). Instead, service requesters are allowed to describe only those parts of a service they are really interested in. Second, text-based and semantic criteria can be combined within a query, while providing imprecise matching capabilities in terms of similarity thresholds, and text-based DoM values referring to the similarity of semantic concepts, which are not considered within other retrieval approaches (e.g., [64, 113, 123]). In order to facilitate user acceptance, SWS2QL makes use of an existing query syntax, namely SPARQL, or rather of an enhancement of SPARQL, which is compatible to the existing SPARQL grammar. Although, some individual query languages designed for specific purposes meet the respective requirements (e.g., [211]), users have to get familiar with the syntax and structure before being able to submit service requests. Finally, SWS2QL does not depend on specific standards, and thus, can be applied to different service description formats, registry standards and matchmaker implementations. For this, a mapping of common service formalisms (i.e., SAWSDL and OWL-S) to a common description format, namely ATSM, is provided, which can be easily enhanced for the integration of further service description standards into service registries. Furthermore, different matchmakers can be specified within a single query in order to account for the specific properties of the service components. Hence, the expressive power of SPARQL is enhanced, since an arbitrary comparison of values is enabled. Although, some approaches also consider the selection of different matchmakers on runtime, this selection cannot be dynamically controlled by a service requester (e.g., [64]).

4.5 Related Work

Similar to service matchmaking, the integration of SWS descriptions in service registries has been examined from the very beginning of SWS research. However, surprisingly little effort has been put into the deployment of alternatives to keyword- and query by example-based query formulation. In the following, a brief overview of service integration approaches into registries will be given. Afterwards, the usage of query languages in SWS research will be examined. The section closes with an overview of these approaches and a comparison with the work at hand.

4.5.1 Integration of Semantic Information Into Service Registries

In another seminal work of Paolucci et al., the authors present the integration of DAML-S profiles in UDDI [200]. The authors propose the mapping of a service profile to UDDI records. If there is no counterpart of a particular profile parameter, a custom-built tModel representing it is generated. For each attribute within the DAML-S profile, a specific UDDI tModel is defined and stored as part of a business service record. Thus, semantic information can be directly accessed by a business service instance. In addition, the DAML-S tModel is defined, which indicates that a given service has a semantic DAML-S description and specifies the URI of the service. A mapping of a DAML-S service grounding to the corresponding binding template is not addressed by the authors.

Besides the DAML-S/UDDI mapping, an external matchmaker architecture is suggested by the authors, which uses DAML ontologies publicly available on the Web for semantic capability matching. The matchmaker described in more detail in Section 3.5 is enhanced by considering preconditions and effects. When service advertisements described by DAML-S profiles are published in UDDI, a UDDI service description is constructed in the UDDI registry, which is associated with the corresponding DAML-S capability description stored in the advertisements database as part of the DAML-S matchmaker architecture. Hence, it is possible to search for services using both, UDDI keyword-based search and the capability matching engine, if requests are specified in the DAML-S format. As result of a matching process, the advertisements and the respective UDDI service records are returned to the requester [200].

Another early approach has been proposed by Akkiraju et al. [5]. Regarding the integration of semantic information using tModels, this work is based on the work presented in [200]. However, while Paolucci

et al. provide an external matchmaker for DAML-S, Akkiraju et al. integrate the matching within the UDDI registry. Therefore, UDDI's `find_service` API is altered so that requests can be enriched with a number of semantically described inputs and outputs. Furthermore, the authors provide an extended matching engine which also allows for inexact matching, i.e., which considers two properties as a match even if only a close relationship between them can be inferred from the respective ontology.

The work of Srinivasan et al. is also based on the work of Paolucci et al. [234]. Here, the authors follow a similar approach for the integration of OWL-S profiles in UDDI. Regarding the deployed matchmaking approach, all known ontology concepts are annotated with a particular DoM at publication time, i.e., for each semantic concept in a new service advertisement, the DoM with all available ontology concepts is computed. This course of action accelerates the actual matchmaking process and can be observed in many of the following approaches.

The work presented by Sivashanmugam et al. primarily focuses on a lightweight integration of semantic annotations in WSDL [231]. This approach has later been picked up in WSDL-S (which has been initiated by the same research team) and SAWSDL. Semantic concepts are referenced by annotating dedicated WSDL elements with corresponding URIs pointing to DAML+OIL ontologies. Apart from this, the authors also provide an approach to integrate the semantically annotated WSDLs in UDDI, which again follows the work of Paolucci et al. In a nutshell, tModels are used to represent the semantic concepts used to annotate operations, inputs, and outputs, which are stored outside the service registry.

The approaches mentioned so far require modifications of the UDDI infrastructure. In contrast, Luo et al. propose a model where the functionalities enabling the usage of semantic search in UDDI need to be installed on client-side [170]. The OWL-S profile and grounding are stored in the UDDI by mapping them to the corresponding data fields of the UDDI records, or the referenced semantic concepts are represented by named tModels. Hence, the semantic information can also be processed by the UDDI search engine. Since the OWL-S grounding corresponds to WSDL documents, it is externally referenced by the UDDI registry analogously to WSDL files. Noteworthy is the authors' recognition of *Anonymous instances*, i.e., composite concepts defined in a service description which are not directly referring a semantic concept in an ontology. Like in [234], reasoning is done at publication time but considers also the already mentioned anonymous instances; matchmaking is based on subsumption matching. Like in the abovementioned approaches, queries are service description-based. Additionally, a Graphical User Interface (GUI) is provided in order to ease the definition of OWL-S profiles.

Primarily aiming at the flexibilization of UDDI's search capabilities, Colgrave et al. propose an enhancement of UDDI, thus allowing to make use of multiple service description languages and corresponding external matching services [64]. The authors aim to overcome the drawbacks of common SWS retrieval approaches, which either make use of external semantic matching facilities separated from UDDI's internal search capabilities or of a single, specific semantic matchmaker which is integrated into UDDI. Therefore, Colgrave et al. propose an enhancement of UDDI's search function allowing for the integration of external matchmakers and their dynamic selection at runtime. In doing so, matching service providers can register their matching services as Web services in UDDI. Service providers and requesters can then publish their external service descriptions and requirements, which are all stored in the UDDI registry as tModels pointing to the external documents via a URL. In order to retrieve suitable services, requesters can formulate `find_tModel()` requests using a new categorization introduced by the authors to indicate their demand for external matching. Appropriate matching engines are then dynamically selected and invoked by the UDDI registry based on predefined selection policies and the desired service description format. In general, more than one matching engine can be selected for each service description format. Although the authors do not make use of a specific query language, three aspects are important to the work at hand. First, the authors' approach is able to cope with various external service descriptions and requirements in order to semantically enhance the search facilities of UDDI registries. Second, external matchmakers can be integrated, so that alternative query and service standards can be processed. Third, the approach allows for the combination of multiple matchmakers for a single query, so that semantic and syntactic matching can be combined.

Kourtesis and Paraskakis propose a combination of SAWSDL, OWL DL, and UDDI (Version 2.0) for semantically enhanced Web service discovery in the *FUSION Semantic Registry* [145]. While this framework does not rely on any specific SWS standard, the reference implementation presented is based on SAWSDL. Neither the UDDI server nor its specification API are altered, but are wrapped in the semantic registry. The publication process is initiated by a query specifying service attributes (e.g., service name, UUID of

the provider) and the URL of the corresponding SAWSDL document. After that, the Semantic Registry SAWSDL parser extracts the URIs of the model references. The information provided by the query and the extracted URIs are then used to create a UDDI service advertisement consisting of a business service UDDI record and referenced tModels, which are used to represent the semantic concepts. At publication time, an Advertisement Functional Profile (AFP) is created from the extracted semantic annotations of the SAWSDL document and added to the OWL knowledge base of the registry.

A service request is expressed as Request Functional Profile (RFP), which will be investigated in the next section. When an AFP is published, a matchmaking/classification against all known RFPs and the new AFP is carried out. References to matching RFPs are then added to the corresponding UDDI business service record of the AFP using tModels. If a URI of a known RFP as part of a service discovery query is submitted to the registry by a service requester, the information from the pre-classification process can be used for service retrieval. In case of an unknown RFP, a matchmaking process has to be performed at runtime in the same way as the pre-classification.

The work of Sivashanmugam et al. is applied in the METEOR-S framework. Here, the *Web Service Discovery Infrastructure* provides the necessary technical means for service discovery on distributed service registries [250]. Similar to the solution presented by Kourtesis and Paraskakis, the service registry is based on UDDI. Again, the UDDI itself is not changed but capsuled by the framework. Service requests are formulated by the required inputs and outputs.

A different approach to the integration of semantic information in service registries has been implemented in PYRAMID-S [211]. Actually, PYRAMID-S is an overlay to service registries which uses a hybrid P2P topology to manage heterogeneous service registries. The aim of the framework is to allow unified Web service publication and discovery, which does not adhere to a particular service registry standard. Furthermore, the authors provide PYRAMID-S WSDL (PS-WSDL), a proprietary SAWSDL-similar semantic extension of WSDL, which is used as the publication format for Web services. Regarding the content of this thesis, the most important aspect of PYRAMID-S is the usage of the Unified Service Query Language (USQL) as service query language [199, 211]. USQL will be further investigated in the next section. As PYRAMID-S facilitates the usage of different service registry standards, it is necessary to define mediators for the designated standards. Mediators for UDDI (based on [63]) and ebXML (based on [58]) have already been defined. For this, a mapping from the PYRAMID-S languages to UDDI and ebXML entities is defined. Concerning PS-WSDL-based service advertisements, these mappings comprise associations with default registry entities (e.g., the `businessService` entity of UDDI or the `Service` class of ebXML) and additional mechanisms for storing semantic information and QoS metadata. In order to store additional information in a UDDI registry, the authors also suggest the application of specific tModels, which are registered once in the registry, in accordance with the approaches mentioned in the last section. Regarding ebXML, additional slots are introduced specifying name/value pairs to add QoS attributes to services and further classification schemes are used [211].

Dogac et al. introduce another approach, which incorporates the integration of OWL ontologies into ebXML registries in order to enhance service discovery. The authors state, that there are generally three ways to add semantics support to the ebXML registry. Basically, these approaches differ in the necessary amount of modifications to the ebXML registry architecture and implementation. The first approach makes use of the existing ebXML RIM classes, or rather the ebXML classification hierarchies and specifies a respective mapping for the OWL constructs, so that the registry does not have to be altered. In doing so, the querying facilities of ebXML can be used by a client to retrieve the semantic information, but the client requires appropriate mechanisms to process the semantics. To overcome this drawback, the authors propose a second approach which makes use of custom-built stored procedures capable to process the OWL constructs. However, in order to provide full reasoning capabilities, a third approach is needed, which considers the integration of rules into the registry leading to changes in the registry architecture.

Since the authors aim is to provide an approach compliant to the existing registry specification, they follow the second approach. For this, they define a mapping of OWL language elements to ebXML class hierarchies, which can be performed automatically from a given OWL ontology by applying the specified transformations. Since the authors are using queries instead of reasoning to obtain knowledge, they make use of OWL Lite ontologies. Concerning the suggested mapping, OWL classes are represented through classification nodes in ebXML, while RDF properties are modeled using ebXML associations. For example, the subclass relationship of two OWL classes is defined by introducing a new association type “subClassOf”. This allows to represent whole OWL class hierarchies through ebXML elements. XSD data

types, which may be part of OWL, are represented by instances of the "ExternalLink" class of ebXML. Finally, stored procedures are defined in order to handle the OWL semantics, e.g., to obtain all the super- or subclasses of a given class. These stored procedures can then be utilized by users in order to retrieve appropriate services that are classified using the OWL classification nodes from the ebXML registry [71]. Again, the details about the query facilities are elaborated in the next section.

There are further approaches to integrate SWS into service registries in general SWS frameworks, with the abovementioned METEOR-S and Web Service Modeling eXecution Environment (WSMX) being the probably most prominent examples. However, in both approaches the actual registry is more a means to an end than in the focus of the work. In an early WSMO Registry Working Draft, UDDI was intended to provide registry functionalities using five new WSMO tModels [106]. However, for WSMX, which is the reference implementation of WSMO, no further information is given, if a particular registry standard has been applied or not. In fact, WSMX's *Resource Manager* is an internal registry; furthermore, it is stated that an ebXML- or UDDI-based registry could be used for WSMX data persistence [60, 101]. In WSMO, services and requests are handled as distinct elements. Requests are modeled using *goals*, which are formal descriptions of objectives. However, goals as well as SWS descriptions are modeled using WSML and are both made up of the same elements: nonfunctional properties, a capability, and a number of interfaces [81]. In the context of query formalisms, WSMO's approach to define service requests does not differ from query by example-based requests as used in the most approaches presented in this section, even though WSMX offers different discovery possibilities based on keywords and the different WSML variants.

4.5.2 Query Formalisms for Semantic Web Services

An early approach to non-query by example-based requests for SWS has been proposed by Bernstein and Klein [28]. Here, *process-based service models* are applied, which are based on a collection of the interlinked subactivities constituting a service (the *process model*). Utilizing process models, the roles, relationships and attributes of service entities are made explicit, which reduces unintended meanings in both, query and service model [28, 127]. For the modeling of services, the authors present a formalization of common process model primitives (e.g., tasks, inputs, outputs), which results in a process model made up from tasks, ports, resources, exceptions, attributes, and values. Besides resources and exceptions, these primitives are also part of common Web service descriptions like WSDL. Tasks correspond to operations, ports to input and output values, and attributes and their values are an integral part of several elements within a service description.

For retrieval purposes, the authors make use of the fact, that the process models can be considered as entity-relationship diagrams and define the PQL. Although PQL does not address semantic annotations of services, process taxonomies may be exploited in order to recognize specialization relationships. Semantic information can be integrated into service queries without additional query syntax. In general, PQL is reminiscent to SPARQL queries, but distinguishes between different element types each exhibiting a different clause syntax. Thus, the query formulation is more difficult in contrast to SPARQL.

Zhuge and Liu present an approach for Web service retrieval based on a *Service Grid*, which represents an orthogonal multi-dimensional service space [260]. The authors focus on a UDDI registry implementation. The Service Grid is used to establish specialization relationships and the degree of similarity between services and between tModels. Thereby, four basic elements of the tModels (operations, category list, identifier list, and keyword set) and three basic elements of the Web services (tModels, category list, keyword set) serve as foundation for the determination of the specialization relationships. In order to detect operation specializations, the inputs and outputs are compared. For the actual retrieval of Web services the Grid Operation Language (GOL) programming environment is proposed, which provides an SQL-like syntax and semantics to specify a service request. GOL can either be used directly by application developers or it can be utilized by users via a GUI containing a GOL query template. Besides the SQL-like commands, the selection of a service or tModel as target, and the choice between local or universal service repositories, boolean condition expressions can be specified [260]. The author's retrieval approach relies on the specification of a known service as source for the query, so that the name of this service is required in order to specify further condition expressions. Such expressions incorporate specialization relationships and the desired similarity value between source and target [260].

Although this approach makes use of semantic information based on specialization relationships between services and tModels, the approach is bound in the first step to a basic keyword-based search similar to the one basically provided by UDDI. For the second step, the actual retrieval of Web services similar to the given example service, the authors do not make use of ontologies to establish semantic relationships. Instead, their comparison is commonly based on the extraction of metainformation of services and tModels and information about operations retrieved from the overview document of the tModels. Hence, the approach mainly performs a text-based comparison of the elements, except for the determination of concept and subconcept relationships of the identifier and category elements within the respective taxonomies.

A related approach to the work presented in Sections 4.3 and 4.4 has been produced by Iqbal et al. [113]. Here, the authors also make use of SPARQL queries in order to retrieve Web services from an ebXML-based service repository. The approach is based on Sbodio and Moulin work presented in [221] where a very similar approach is used for OWL-S.

SPARQL is used for two reasons: On the one hand to constitute user requests and on the other hand to formalize service pre- and postconditions. More precisely, user goals are specified in the form of SPARQL `ASK` queries, which have to be fulfilled by a matching service, and the SPARQL `CONSTRUCT` query form is used for the representation of a service result including its pre- and postconditions. In this context, user goals describe the state of a service after its successful execution. The matchmaking process proceeds as follows: First, an initialization phase is passed, in which the knowledge base (e.g., available service descriptions, ontologies, the user goal) is loaded. Second, the SPARQL `CONSTRUCT` query is executed over the knowledge base, resulting in an RDF graph describing the effects of a service's execution. If the resulting RDF graph has solutions for the user goal, i.e., the `ASK` query, it can be inferred that the respective service is able to fulfill the desired goal. If no matching service can be found, the matching engine is also able to perform relaxed matching, i.e., conditions are progressively dropped in order to find a more generic solution.

The approach has been prototypically implemented by the authors using ebXML. The registry is used to store the SAWSDL-based service descriptions, while the SPARQL-based conditions are stored separately in the repository infrastructure. For the integration of the SPARQL-based pre- and postconditions, the `modelReference` attribute of SAWSDL is used, which points to the URI of the conditions. To interlink the SAWSDL-based service descriptions with the SPARQL-based conditions, an additional slot is introduced for the registry objects containing the URIs. Concerning the query capabilities, the authors make use of the JAXR API provided by ebXML. The authors state that their ebXML-based service repository does not (yet) allow to query for the integrated semantic metadata.

Nevertheless, the approach by Iqbal et al. addresses some noteworthy aspects with respect to the work at hand. Basically, the authors propose SPARQL as query language, which represents a common query standard, and thus, facilitates user acceptance. In addition, the authors account for flexible matching, i.e., the retrieval of services only satisfies parts of a user request. Although the details of the implementation are not stated explicitly, the authors suggest to store the semantically enhanced service descriptions within the ebXML infrastructure and indicate a mechanism to reference additional semantic information in form of SPARQL-based conditions. Finally, it should be noted, as also stated by the authors, that the application of SAWSDL allows for various DoMs, e.g., interface level matching can be performed using the associated classification information, and operation level discovery can be accomplished based on the annotations of the inputs, outputs, and the specification of pre- and postconditions.

Matchmaking in the already mentioned FUSION Semantic Registry [145] is based on AFPs and RFPs which are essentially OWL classes that represent the category, input, and output of a particular service advertisement or service request. While AFPs are automatically generated from SAWSDLs registered in FUSION's semantic registry, the latter have to be defined by service requesters [146]. This makes RFPs a distinct ontology-based query language for this framework. As an RFP essentially abstracts a SAWSDL-based service description, it might be easier to define for a service requester but does not comprehend any additional information like ranges. Thus, the additional benefit of RFPs is questionable regarding query formulation.

USQL, which has been mentioned as part of the PYRAMID-S framework in Section 4.5.1, is an XML-based query language which aims at the discovery of Web, P2P, and Grid services [199, 211]. The authors' work aims at unified, behavior-based service queries which can be modeled using a visual query modeler [211]. Apart from searching for service capabilities, non-functional properties like QoS are also

regarded. USQL is not based on a certain service syntax but applies an abstract viewpoint of service properties. Thus, a USQL search engine needs to map queries to request formats supported by service registries like UDDI, ebXML, or JXTA (a P2P infrastructure which can be queried for P2P services); the integration of USQL in a service publication framework is presented in [211] (cp. Section 4.5.1).

USQL comprises the usage of semantic annotations in queries by using the attribute "ontReference" which references a semantic concept defined in a formal ontology. This attribute applies to capabilities, inputs, outputs, and QoS properties of a query. USQL also supports the definition of *operators*, i.e., ranges for the three types of requirements. Concerning the syntactic elements, the specified operators include exact, wildcard, and exclusionary modifiers. The semantic operators are similar to the classic DoM values exact, plugin, and subsumes. In addition, a sibling operator can be specified, which addresses concepts sharing the same parent concept. The QoS requirements are not further discussed in the following, since QoS parameters are not considered in the work at hand.

As a result of USQL's abstract viewpoint of service properties, the mapping between a query and a certain service standard is not always clearly defined. For example, it is possible to annotate inputs and outputs in a query with ontology references. However, in SAWSDL/WSDL 2.0, model references (cp. Appendix A.1.1) are not defined for inputs and outputs. There are further examples, which show how only parts of the USQL search criteria are mapped by the USQL search engine to the various service description formats. However, this is a characteristic problem if abstracting from a certain service standard. Even so, Pantazoglou and Tsalgatidou follow a very promising approach with USQL.

Compared to the solutions presented in this thesis, USQL was created with a different basic idea in mind. We want to use a lightweight approach to service query formulation. Hence, it was the aim to make use of an already existing query syntax well-known in the corresponding community instead of building another query language. Both approaches have their right to exist, and it remains to be seen if users will accept another distinct language like USQL or want an enhancement of an already known query syntax like SPARQL. Nevertheless, USQL comes closest to the work at hand.

The approach proposed by Dogac et al. mentioned in the last section deals with the integration of OWL ontologies into ebXML in order to improve service discovery [70, 71]. A user can then make use of the additional semantic information by utilizing the stored procedures provided by the authors for this purpose or through the standard SQL query interface provided by the ebXML registry [71]. Concerning the latter, an average user being not familiar with the database scheme will most likely be already overstrained by a trivial SQL-based query, since the database scheme of ebXML is very complex. Queries exhibit a high degree of nesting and are not very intuitive. Due to the complexity of the SQL queries, the authors provide specific stored procedures that are able to process the semantics, e.g., to recognize the predefined relationships between the semantic concepts [71]. However, these stored procedures do not provide flexibility with respect to individual arguments. Furthermore, their name and the required parameters have to be known in advance by users or clients in order to invoke them. To be able to cope with the query mechanisms in a more comfortable way, a GUI is typically provided by ebXML registries in addition to a programmatic query interface. Such a visual query tool is also provided in this solution, in which the classification hierarchies of ebXML are depicted graphically.

Summarizing, the work of Dogac et al. does not account for inferred semantic relationships, since it relies on querying predefined semantic hierarchies. Although the authors' approach is based on an existing query syntax, namely SQL, the underlying database scheme is highly complex. Hence, the provided stored procedures seem to be more applicable, but they do not offer maximum query flexibility. In contrast to the work of Dogac et al., the thesis at hand aims for integrating reasoning capabilities into registries and for providing a more flexible query approach, which considers the generic properties of all services, instead of a specific service type.

Apart from the already mentioned work by Iqbal et al., the usage of SPARQL with respect to service discovery has been proposed by Kiefer and Bernstein. The authors apply SPARQL-based queries to query services semantically described in OWL-S [123]. The focus lies on the integration of syntax-based similarity metrics into SPARQL. The authors propose iSPARQL, which makes it possible to phrase *imprecise* SPARQL queries. iSPARQL is based on the calculation of a similarity score which is a real number. An overall similarity score can be made up from different (weighted) individual similarity scores. For example, it is possible to determine similarity scores based on well-known methods from IR like TF-IDF or *Levenshtein string similarity* [14, 162]. For each similarity method, a minimum similarity score is integrated in an iSPARQL query. Although proposed for Semantic Web resources in general [124, 125], the

application of iSPARQL for SWS has been explicitly considered [123]. A more detailed discussion of the actual matchmaking approach can be found in Section 3.5.

In order to integrate imprecise matching capabilities into SPARQL, the authors present three extensions of SPARQL. The first extension makes use of so-called virtual triple patterns, which establish virtual relations between resources. For this, so-called *magic properties*[4] are used. These property functions allow for the external matching of triple patterns using customized similarity functions, instead of matching them against the underlying ontology graph. An external matching request is indicated by adding a prefix with a special name to the predicate of a triple pattern. The object of the triple pattern contains the arguments, which are passed to the external similarity function. After the comparison, the computed value is assigned to the subject variable of the triple pattern. Since the similarity between the resources is only based on a virtual relationship, i.e., the relation is not part of the underlying ontology graph, this approach was called *virtual* triple approach [123, 125]. The use of virtual triple patterns is notified to the query processor by introducing a new `SimilarityBlockPattern` to the SPARQL grammar, which starts with the keyword `IMPRECISE`, followed by a number of virtual triple patterns and may contain optional `FILTER` statements that are already part of the official SPARQL grammar [125].

To the authors opinion, utilizing virtual triple patterns bears the following advantages: First, various external similarity measures can be applied, which can be combined to individual similarity strategies. Second, assigning similarity values to variables allows for their reuse within further query statements, e.g., for aggregation or ranking purposes. Nevertheless, this approach also exhibits some shortcomings. An extension of the SPARQL grammar and a corresponding modification of the query engine is required. However, the query engine of *Jena* already allows for the use of property functions [125]. A second extension of SPARQL as proposed by Kiefer et al. requires no extension of the SPARQL grammar [125]. Instead, the semantics of the SPARQL `FILTER` expression is extended to account for extension functions for the computation of individual similarity scores. The authors state that although this approach requires no extensions of the existing SPARQL specification, a query engine must be able to interpret the references to the external similarity measures. Furthermore, they declare, that since the similarity scores cannot be assigned to variables within a `FILTER` expression, they cannot be reused within other query statements for aggregation and ranking. Hence, aggregations have to be performed inside the `FILTER` expression, so that the complexity of the `FILTER` expressions increases. Finally, a third approach is suggested by Kiefer et al., which adds new solution modifiers to the existing SPARQL grammar [125]. However, this approach is not further considered by the authors, since it does not provide any benefits in comparison to the first two approaches. In addition, the authors state some shortcomings: Solution modifiers cannot introduce new variables or assign values to existing variables. Thus, the resulting similarity scores cannot be returned to the user. Furthermore, the authors declare that data constraints should be part of the `WHERE`-clause and that solution modifiers are not intended to retrieve data from the underlying ontology, but solely to handle the result variables.

In their work regarding the usage of iSPARQL in service discovery, Kiefer et al. make use of the virtual triple pattern approach and OWL-S-based service descriptions, but the authors point out that their approach is also applicable to other service description formats [123]. Basically, it can be stated that the virtual triple and the extension function approach are both very promising, since they rely on a common query syntax, namely SPARQL. In addition, both approaches allow for individual similarity computations between resources, and the desired matchmaking function can even be stated within a service query. Furthermore, similarity thresholds can be specified by a service requester. Finally, both approaches represent a reasonable extension of the SPARQL grammar and/or its semantics, in contrast to the last approach suggested, the declaration of additional solution modifiers, which are only intended to modify an existing result set, instead of processing additional queries.

Lamparter and Ankolekar present an approach for automated service discovery, that accounts for service offers providing various configurations of a service and according configurable service requests [159]. Furthermore, they propose a service selection algorithm, which performs a ranking of the service offers based on the requester's preferences. To express service requests, the authors make use of SPARQL queries. Similar to service offers, configurations within service requests are expressed by defining an additive scoring function that assigns scores to the desired configurations. In general, a SPARQL query for a configurable Web service consists of three parts. The first part contains the information required for functional matching, i.e., the inputs and outputs. In the second part, constraints are added to the

[4] http://jena.sourceforge.net/ARQ/extension.html#propertyFunctions

query by specifying boolean, attribute-based filter conditions. Finally, within the third part, preferences are included into the query for ranking purposes. This allows the ranking to be accomplished by the server. Since the inference engine utilized by the authors permits the formulation of so-called preference queries realized by built-in predicates within a SPARQL query, the authors extend the SPARQL syntax by the `EVALUATE` keyword and introduce the *pbF* predicate. The predicate indicates that the arguments, which represent scoring values, and the associated attribute value are passed to a function. This function computes the final score of the attribute value. As result of a query, the final ranking of the service offers is revealed to the user. To facilitate the composition of SPARQL queries, a graphical tool is provided by the authors [159].

Although service configurations are not considered within the thesis at hand, the approach of Lamparter and Ankolekar contains some aspects worth mentioning with respect to the work at hand. Generally, the authors make also use of SPARQL as query language based on an abstract Web service offer ontology. This ontology is expressed in terms of a UML class diagram, which is a common way for representing ontologies in an informal way [37, 118]. Furthermore, they propose a way of specifying preferences directly within a query, which are comparable to DoM values. In comparison to the iSPARQL strategies introduced by Kiefer et al., Lamparter and Ankolekar also apply built-in predicates to refer to external functions, which resembles the virtual triple approach. In addition, they also utilize `FILTER` expressions, but in contrast to the extension function approach of Kiefer et al., they do not extend the `FILTER` expression semantics [159]. Nevertheless, it can be stated that Lamparter and Ankolekar follow a combined approach by specifying attribute filters and preferences.

4.5.3 Overview and Comparison

Table 4.2 shows an overview of the related work presented in the last two sections. The focus is on approaches which depart from query by example-based service requests. The second column presents the features of each approach, making use of the following symbols.

C Hybrid search (syntax and semantics)	DM Different matchmakers
R Similarity ranges	DS Different SWS formalisms
Q Reuse of existing query syntax	RS Different registry standards

The features are based on the requirements defined in Section 4.2 and already assessed for the solutions at hand in Sections 4.3.6 and 4.4.5. The left row aims on generic query language properties, while the right row examines if a framework is independent from certain technological constraints. The former criteria comprise the support for combining syntax-based and semantic criteria within a single query (C), the capability of defining a threshold respectively range (R), and the utilization of an existing query syntax (Q). Concerning later criteria, the approaches are analyzed with respect to their applicability to different matchmakers (DM), multiple service description standards (DS), or different registry standards (RS). The results of the comparison are listed in Table 4.2 and elaborated on in the following. A criterion is put in parentheses if it cannot be controlled by the service requester, i.e., the criterion is automatically managed by the respective approach, or the criterion is only supported with limitations.

If comparing the work conducted in Sections 4.3 and 4.4 with each other, the differences are obvious: While the solution conducted for SPARQL, SAWSDL, and UDDI is lightweight, it does not provide sophisticated ways to customize the query formulation and execution, as such features make it necessary to define an abstract service and query model as foundation. In contrast, SWS2QL is based on such models and hence able to provide a more sophisticated query formalism. Furthermore, the query statements formulated in SWS2QL are much more user-friendly and better arranged than the original SPARQL queries applied in Section 4.3. Hence, SWS2QL is a unified, holistic approach to query formulation, while the other approach is primarily technology-driven and restricted to a particular application scenario.

Not every approach presented in Table 4.2 concerns the work at hand to the same degree. To start with, the approaches presented by Colgrave et al. and Kourtesis and Paraskakis do not really make use of a query language, but provide another alternative form to query formulation. Klein and Bernstein require services to constitute process models [28, 127]. The use case of the work of Zhuge and Liu is constrained by the need to establish a specific data structure, i.e., the so-called Service Grid [260]. Dogac et al.

Table 4.2: Comparison of Related Work with the Presented Solutions

	Features	Query Syntax	Semantic Information	Web Service Standard	Registry Standard
Kourtesis and Paraskakis [145, 146]	C, (R)	Query & OWL DL	OWL DL	SAWSDL	UDDI
Klein and Bernstein [28, 127]	C, R	PQL	process taxonomies	process models	undefined
Iqbal et al., Sbodio and Moulin [113, 221]	C, Q, (DS)	SPARQL	OWL, SPARQL +OWL-S	SAWSDL, OWL-S	(ebXML)
Dogac et al. [70, 71]	(C), (R), Q	SQL, stored procedures	OWL	WSDL	ebXML
Zhuge and Liu [260]	(C), R, (Q)	SQL-like	Service Grid	WSDL	UDDI
Lamparter and Ankolekar [159]	C, (R), Q	SPARQL	OWL	WSDL, OWL DL + SWRL, WSDL	undefined
Colgrave et al. [64]	C, (DM), DS	tModels	undefined	tModels	UDDI
Pantazoglou and Tsalgatidou [199, 211]	C, R, DS, RS	USQL (XML-based)	OWL	PS-WSDL	UDDI, ebXML
Section 4.3	C, (R), Q, (DM), DS	SPARQL	OWL DL	SAWSDL	UDDI
Kiefer et al. [123, 125]	C, R, Q, (DS)	iSPARQL	OWL	OWL-S	undefined
Section 4.4	C, R, Q, DM, DS, RS	SWS2QL (SPARQL-based)	OWL DL	OWL-S, SAWSDL, ATSM	ebXML, (UDDI)

enable SWS discovery by integrating OWL-based ontologies into ebXML [70, 71]. Unfortunately, this solely permits the retrieval of knowledge based on querying predefined semantic hierarchies, instead of inferring semantic relationships. Although the authors utilize the existing query mechanisms of ebXML, the SQL-based queries exhibit a high level of complexity whereas the stored procedures reveal a lack of flexibility.

In general, the approach proposed by Pantazoglou and Tsalgatidou is closest to the work at hand [199, 211]. USQL and SWS2QL both make use of a conceptual service model to describe the content of a query in an abstract way. However, USQL represents a more heavyweight approach than SWS2QL, since it accounts for several service types by default, not only SWS. Basically, the abstract query model, which provides a structural foundation for SWS2QL, resembles an USQL request. But with regard to search criteria, the abstract model provides a more generic way of specifying elements, which allows for a simple adoption of the model and respectively SWS2QL when facing different search criteria, i.e., if the underlying content model (here: an extension of ATSM) is changed. Within USQL, the elements are explicitly stated and also exhibit specific properties, which inhibits an easy adaptation to changing search

criteria. Nevertheless, USQL also allows for the specification of text-based and semantic search criteria in the form of references to concepts defined in some formal ontology. Regarding similarity ranges, USQL also permits the specification of a minimum degree of similarity on a global level, i.e., for the overall request, and also for single properties just like SWS2QL. Unfortunately, no syntactic expressions are stated by the authors, how thresholds are assigned to the properties. Another drawback of USQL is the missing opportunity to declare different matchmakers within a query. Matchmakers or rather mediators can only be indirectly selected via the specification of target registries. Hence, no multiple matchmaker selection is supported for a single query. In addition, USQL does not make use of a standard query language, which forces service requesters to get familiar with the particularities of each service element to be specified within a query. But, like SWS2QL, USQL also provides a mapping to the individual query formats of the different registries. Concerning the integration of service descriptions into different registries, the authors make use of a new description format, PS-WSDL. In doing so, an individual mapping of the components and properties contained within a PS-WSDL file is defined for each registry standard. However, service providers are forced to express their service offers using PS-WSDL, since the authors do not provide a mapping from other service description formats to PS-WSDL. In addition, the use of PS-WSDL demands an enhancement of each registry mapping, when new service properties should be integrated. In contrast, ATSM represents a more flexible approach, since it represents a common description format and the result of a mapping from individual service description standards. So, users can describe their service offers using an arbitrary description format. Even the registry mapping stays the same when new properties are added, since only the ATSM model itself has to be enhanced.

The work by Kiefer et al., iSPARQL, is also close to the thesis at hand, since the authors propose the integration of imprecise matching capabilities into SPARQL for hybrid matchmaking [123, 125]. As the approach aims on imprecise matching, matchmaking is conducted using IR-based similarity values, which are applied to service attributes (e.g., name, description) or sets of concept names resulting from the unfolded input and output concepts of a given service. Hence, no subsumption-based matching functions are utilized. Therefore, iSPARQL only permits the specification of numerical similarity thresholds referring to the overall similarity of attributes or concept name sets. Additionally, non-numerical DoM values can be specified in SWS2QL in order to determine precisely, if an output or input of a service offer can be used instead of the respective input or output specified within a service request. Contrary to iSPARQL, SWS2QL allows for the specification of similarity measures or rather matchmakers for a whole query section. This permits the combination of similarity values resulting from query sections instead of aggregating similarity values of single comparisons, and thus, increases usability. Finally, neither an integration mechanism of iSPARQL nor of service description formats into service registries is specified. Although, the authors state that their approach could also be applied to other service description standards, they only make use of OWL-S.

Apart from Kiefer et al., Iqbal et al. and Lamparter and Ankolekar also take SPARQL as SWS query language [113, 159, 221]. Iqbal et al. apply SPARQL `ASK` and `CONSTRUCT` queries to retrieve SAWSDL- respectively OWL-S-based service descriptions. These service descriptions are enhanced with pre- and postconditions based on SPARQL and OWL-S. In contrast to SWS2QL, the authors focus on preconditions, result conditions and effects of a service. Besides the provision of a common query syntax, this approach also permits relaxed service matching, i.e., conditions can be progressively dropped if necessary. However, this approach does not allow for the specification of similarity thresholds. As it was stated before, the authors make use of ebXML as service registry standard. However, they do not state any details on the integration of Web service standards into ebXML. Lamparter and Ankolekar suggest an approach for automated discovery and selection of configurable Web services. For this, service offers are enhanced with additional configuration information and specified using OWL DL and DL-safe SWRL rules. Service capabilities only comprise inputs, outputs, and attributes referring to configurations; service requests are expressed using SPARQL. Similar to SWS2QL, a subsumption-based matching of the specified input and output concepts is performed in the first step. In contrast to SWS2QL, no similarity thresholds can be defined. However, the authors also allow for the specification of preferences in terms of numerical weights and filters for the desired attributes of a configurable service.

4.6 Conclusions

Service request formulation is dominated by keyword- and query by example-based approaches. Even though a few approaches for the usage of query languages in service discovery have been presented, these are usually restricted to particular service formalisms, do not make use of an already existing query language, or are subject to other restrictions. In this chapter, two distinct approaches to query formulation based on SPARQL, the standard query language of the Semantic Web, have been presented. Both approaches aim to provide an intuitive, user-friendly query interface and overcome the restrictions of usually applied query formalisms, like, e.g., the missing definition of thresholds, parameterization of matchmakers, and separate handling of syntax-based query statements.

The first approach benefits from the already existing W3C SAWSDL and WSDL to RDF mappings as well as existing approaches to integrate WSDL-based service descriptions into UDDI. Thresholds are defined for single service components using the actual query parameters. This way, a lightweight implementation is enabled, which can easily be integrated into an existing software landscape made up from a standard UDDI and an RDF framework like, e.g., Jena. However, the lightweight approach also includes some drawbacks. As it is making use of already existing technologies and standards, the features of this approach are restricted by these: First, the processing of thresholds is not part of the query model but thresholds have to be explicitly considered by the query processor. Second, the queries are not well-arranged due to the official mapping from WSDL to RDF and are furthermore restricted by this mapping, as there is no mapping from another SWS formalism to the same RDF structure. Third, the assignment of different semantic matchmakers is not recognized or remains inexplicit. Finally, the solution is restricted to UDDI.

In contrast, the second approach provides a more extensive approach, which is applied to ebXML-based service registries. Therefore, we present SWS2QL, an extension to SPARQL, which integrates thresholds and the possibility to define and configure different matchmakers in a query. As SWS2QL is independent from a certain SWS formalism, matchmaking approach, or registry standard, the language provides a unified querying approach for Web services. SWS2QL is implemented in a general framework based on ebXML, which features the integration of SWS descriptions and different service matchmakers into this registry standard. The application of extended SPARQL queries as proposed in this thesis presents a unified approach to service query formulation, which exceeds the current state-of-the-art while applying an already known query syntax.

Regarding the hypotheses presented at the beginning of this chapter, we can make the following statements:

H4: The slightly extended version of SPARQL applied in Section 4.3 possesses some major shortcomings: As SPARQL represents a query language for RDF, but is not intended to be applied to SAWSDL, it misses certain particularities required for flexible and precise SWS retrieval. However, SPARQL is an appropriate foundation for a more specialized query language for SWS, namely SWS2QL as presented in Section 4.4. If comparing the features of SWS2QL with those of commonly applied approaches or the integration of SPARQL in UDDI as presented in Section 4.3, SWS2QL exhibits a higher degree of user-friendliness as well as means to define a fine-grained definition of service requirements while offering common query language features as e.g., the possibility to define thresholds, and a query syntax mostly well-known to users of SPARQL.

H5: Despite the fact that the proof of concept implementation presented in Section 4.4 is conducted for the ebXML registry standard and SAWSDL, the query formalism is per se transferable to UDDI or other, proprietary service registry specifications. This applies also to other Web service and query formalisms due to the decoupling of the service query model and query language from the integration into the service registry and the decoupling of the service data model from the Web service formalism. Even though this increases the number of service and query formalisms, the usage of abstract service data and query models is necessary in order to establish a standard-independent view. Therefore, the usage of abstract models eases the actual integration of a possible unified service query language like SWS2QL.

There are possible extensions of the work conducted: To start with, the depiction of query results is usually done by presenting a list of suitable services, which are sorted based on the similarity to the query. A sophisticated visualization of matching results in terms of a graphical output of the single

service components, which are augmented by or colored based on the single similarity values, would be certainly helpful. Here, work conducted in the field of service composition, e.g., by Bodenstaff et al. [31], could be used as a foundation.

Another major aspect of the future work could be the transfer of the results presented in this chapter to future SPARQL versions. Parallel to the work at hand, W3C's SPARQL Working Group has released the first working drafts of SPARQL 1.1, which integrates new operators, like `MINUS`, adding support to reduce a result set represented by one operator by results from a second operator [126]. In an extension of SWS2QL, new features of SPARQL should be regarded and employed.

Part III
Finale

5 Conclusions and Outlook

In this chapter, the fundamental findings of this thesis will be recapitulated and an outlook on possible future work will be given.

5.1 Conclusions

In this thesis, two major aspects of Web service discovery based on semantic information have been observed, namely service matchmaking and query formulation. The discovery of Web services is one of the basic operations and hence a matter of great concern in Service-oriented Computing. The outcome of the discovery process is constrained by the ability of service providers to describe their service offers and the ability of service requesters to define their requirements towards services as well as the effectiveness of the service matchmaker, i.e., an algorithm that finds the best service offers based on a service request. As syntactic information is often not sufficient to describe service capabilities, the integration of semantics into Web service descriptions is deemed as a major challenge in Web service discovery. Semantic information can be used in service offers as well as service requests. In order to benefit from this information, matchmakers need to be able to process it. While matchmaking based on explicit semantics leads to a higher grade of reliability of discovery results, state-of-the-art matchmakers also make use of implicit semantics in order to improve matchmaking results.

Within this thesis, two matchmaking approaches for SAWSDL have been developed, implemented, and evaluated. The matchmakers aim at capability matching and take into account information from the different service abstraction levels of SAWSDL.

The first matchmaker, LOG4SWS.KOM, is based on "classic" subsumption matching and provides an innovative adaptation mechanism with regard to differing basic assumptions concerning the semantic concepts applied in service and query descriptions. Due to this, it is possible to automatically adapt LOG4SWS.KOM to different service domains and domain ontologies. Adaptation is done regarding the ranking and weighting of the distinct DoMs detected during matchmaking, but is independently conducted for different service abstraction levels. Thus, LOG4SWS.KOM is able to handle differing basic assumptions on these levels, i.e., what a semantic annotation for an interface, operation, or message parameter actually means. As not every SWS is perfectly semantically described, LOG4SWS.KOM also provides a linguistic-based fallback method to be used if a service component is not semantically annotated or the semantic information cannot be processed. As the similarity measure computed by LOG4SWS.KOM is a numerical value, it can be easily combined with other similarity metrics. Thus, LOG4SWS.KOM provides the first well-motivated mapping from distinct subsumption DoMs to numerical values.

While the second matchmaker developed in the context of this thesis, COV4SWS.KOM, shares a number of features with LOG4SWS.KOM, there are two essential distinctions between the two matchmakers. First, the similarity metrics applied in COV4SWS.KOM originate from the field of semantic relatedness in ontologies yielding more fine-grained similarity values. Furthermore, adaptive weighting of different service abstraction levels is introduced – based on the quality and usefulness of semantic and syntactic information on the different levels. Through the evaluation of COV4SWS.KOM, we were able to show that existing similarity metrics determining semantic relatedness are a well-suited alternative to logic-based similarity determination and that the proposed adaptation approach is likewise producing very good matchmaking results.

To the best of our knowledge, LOG4SWS.KOM and COV4SWS.KOM provide the best evaluation results regarding Information Retrieval metrics like Average Precision and R-Precision of any SAWSDL matchmaker, so far. While evaluation results for the different variants and versions of COV4SWS.KOM are generally not as good as those of LOG4SWS.KOM, there is one important exception – namely Variant 4a, which makes use of the semantic relatedness metric by Resnik and OLS-based service abstraction level weighting and provides the best IR results in this thesis, and therefore, for any SAWSDL matchmaker.

Through the application of LOG4SWS.KOM and COV4SWS.KOM, we were able to show that (self-) adaptation of service matchmakers has not to be necessarily conducted regarding the combination of

different similarity metrics as done in the related work. Instead, adaptation can also be applied regarding the weighting of service abstraction levels (COV4SWS.KOM), or differing basic assumptions of semantic concepts and their relationships in ontologies and semantic service annotations (LOG4SWS.KOM). Both adaptation mechanisms can be used in order to apply the matchmakers to certain service domains – if services in a domain share a similar degree of descriptions' usefulness on the different service abstraction levels, COV4SWS.KOM is a valid choice. If the services are semantically annotated following the same principles, LOG4SWS.KOM is well-suited to be applied. In both cases, the matchmakers provide a powerful basic functionality, which could be applied to industry-specific marketplaces in the "Internet of Services". As it was mentioned in Section 3.4.3, different domains make use of ontologies which may follow different design principles. Hence, different configurations of the matchmakers could be used in order to distinguish between service domains even in the same service marketplace.

As a final aspect, the matchmakers provide not only similarity values between complete service descriptions, but also for single service components. This way, they can be applied as underlying matchmakers in query processing as presented in the following.

The employment of query language-based service requests has been the second focus of this thesis. Based on an extensive analysis of requirements a query formalism for SWS should address, two distinct approaches to query formulation for SWS have been proposed. The first is based on already existing technologies and standards, i.e., (slightly extended) SPARQL as query language, the official W3C SAWSDL and WSDL 2.0 to RDF mappings as foundation for the transformation of SAWSDL-based service descriptions to RDF, and existing integration approaches for UDDI. Using a mapping from SAWSDL to RDF, it is possible to integrate RDF-based service descriptions into UDDI. In a prototypical implementation, LOG4SWS.KOM is used in order to perform matchmaking if ranges have been defined. If only exact matches are desired, matchmaking can be directly performed using a SPARQL processor. The aim of this UDDI-based prototype was to show how a lightweight extension of a service registry's search capabilities can be easily achieved. The evaluation of this approach has shown that the reuse of already existing technologies is suitable to facilitate SPARQL as query language in UDDI, but there are certain restrictions, which make this approach not very user-friendly. Most notably, this addresses customization of matchmaking and the transferability to and incorporation of other Web service standards.

These shortcomings are addressed by the second solution, which represents a holistic approach to query formulation for SWS. As this approach is more sophisticated, the expressive power of SPARQL was not sufficient. Hence, it was necessary to enhance SPARQL by new features aiming at SWS discovery. This is done using an abstract query model, which refers to an abstract data model for services. The actual query language, SWS2QL, is an enhancement of SPARQL based on these abstract models. SWS2QL is compatible with the existing SPARQL grammar, makes use of existing extension principles, and minimizes the amount and complexity of further rules. SWS2QL allows a service requester to address different service abstraction levels, incorporate and parameterize matchmakers, define similarity thresholds, etc. Due to the abstract models SWS2QL relies on, the query language may be easily transferred to different service registry standards. Nevertheless, the current proof of concept implementation has been conducted for ebXML Registry, a primarily commercially applied service registry standard.

From a more practical viewpoint, SWS2QL offers a powerful search facility for the "Internet of Services" if the language and the underlying service data and query models are applied in upcoming service discovery frameworks. As SWS2QL is SPARQL-based, the language is usable by a large community. In the opinion of the author, the reuse of an already existing query syntax well-known to the corresponding community is a crucial factor necessary to fulfill in order to assure the acceptance of a query language. To the best of our knowledge, SWS2QL is the first query language to explicitly address and fulfill this requirement. Another aspect of the proof of concept implementation with practical impact is the integration of semantic service matching in ebXML Registry, which is a prerequisite for advancing the acceptance of SWS in the software industry. This includes particularly the definition of an interface for SWS matchmakers and the integration of SWS descriptions in ebXML Registry.

5.2 Outlook

If regarding the research agenda for Service-oriented Computing depicted in Figure 1.1 in Chapter 1, this thesis has primarily addressed functionalities from the bottom layer, i.e., basic operations. Even though many approaches have been proposed to service discovery and service publication, this thesis presents

innovative approaches to service matchmaking and service query formulation. Nevertheless, there remain some open questions worth regarding:

To start with, service matchmaking as presented in LOG4SWS.KOM and COV4SWS.KOM could be applied to other service formalisms, like WSMO or OWL-S. The latter can be easily done, as OWL-S has already been considered in ATSM. There are further, REST-based, service formalisms, as the Web Application Description Language (WADL) or SA-REST [100, 228], which have not been regarded in matchmaking research so far and would also constitute an interesting area of application for the matchmakers proposed in this thesis. Furthermore, the recently released Universal Service Description Language (USDL) provides an extensive meta-model integrating business, technical, and operational aspects for services on the "Internet of Services" [50]. Here, it would be interesting to examine if such a comprehensive model requires a multi-step matchmaking approach instead of the single-step capability matching usually applied today. A second aspect of service matchmaking that has so far only been paid little attention to, is the scalability of Web service matchmaking. In an "Internet of Services", where billions of services are provided and consumed on the Web, the current runtime performance of COV4SWS.KOM, LOG4SWS.KOM, and competing approaches will not be sufficient. Here, the usage of heuristics and methods to scale down the search-space are promising. Third, the adaptation of service matchmaking results may not only be applicable for the combination of different similarity metrics as applied in the related work, the derivation of ranking and weighting for distinct Degrees of Match as applied in LOG4SWS.KOM, or the weighting of different service abstraction levels as applied in COV4SWS.KOM, but also for the ranking of matchmaking results based on past service invocations. Fourth, we assume so far that semantic annotations are correct and consistent. As Fensel points out, this is not necessarily true [180]. Furthermore, ontologies are not necessarily static but might evolve over time [99]. Hence, it might be useful to explicitly consider matchmaking with inconsistencies and multiple ontology versions.

If ascending the pyramid from Figure 1.1, and regarding "service selection", service matchmaking as applied in this thesis is restricted to functional properties of services but does not include non-functional properties like QoS. However, the integration of QoS aspects into service selection is an elementary necessity. With our work on *WS-Re2Policy 2.0* [223], we have already proposed an approach on how to integrate semantic annotations of QoS aspects as well as deviation handling. In the future, such semantic information will play an important role in the research group's work on QoS management, adaptation and monitoring mechanisms, and self-organization as initially shown in [222].

During the extensive evaluation done in Chapter 3, we noticed that common evaluation approaches used for Web service matchmakers are quite modest. Most importantly, evaluation results are frequently judged based on rather few metrics and are not always reproducible. In contrast, LOG4SWS.KOM and COV4SWS.KOM have been tested in various variants and versions. This does not only permit to apply different variants of these matchmakers to different service domains, but is also necessary for other researchers to follow, reproduce, and enhance the work at hand. Based on our proposal, the next version of the Semantic Web Service Matchmaker Evaluation Environment (SME2), which has been used as evaluation framework in this thesis (cp. Appendix B.2.1), will include further evaluation metrics and the possibility to automatically conduct several evaluation results for different versions/variants of a matchmaker. Currently, we participate in the research community's efforts towards a more comprehensive benchmarking framework with metrics that can be used to measure the performance/scalability, correctness, etc. of SWS discovery and matchmakers. Even though, the evaluation approach and test data set applied in this thesis was suitable to perform a quantitative evaluation of LOG4SWS.KOM and COV4SWS.KOM, further evaluations regarding the *equivalence*, *scope*, and *interface compatibility*, which are currently mixed to some degree, might be possible in the future [154].

Regarding the integration of query languages in service registry standards, the work conducted in Section 4.4 could also be adopted to UDDI. This has already been arranged for by the usage of the abstract data and query models, which could be applied to further registry standards, too. As another aspect, the matchmaking as well as query facilities, which are integrated into ebXML Registry at the moment, could be externalized from the actual registry and provided as external components. This would lead to an even more flexible service discovery framework. Apart from the actual registry standard or matchmaking method applied, the depiction of matchmaking and query results needs to be improved: Usually, a service requester receives a ranked list of matching services, which might be augmented by the actual similarity values computed for each service. Service-savvy requesters, in particular, will benefit from an alternative representation of matchmaking results, e.g., a visual description of which service

components meet the requirements of the requester and which do not. As another aspect affecting query formulation, the currently conducted work on SPARQL 1.1, which will integrate further operators into the SPARQL query language, needs to be regarded in the future.

One particular application area of services, which will most likely play a very important role in the near future, is *cloud computing*. In order to realize the vision of a global cloud computing service market, appropriate service discovery mechanisms are required, which incorporate dynamic negotiation of service level agreements. In doing so, techniques and mechanisms from the field of SWS discovery can be adjusted to the needs of services in the cloud. Especially if regarding the invocation of services on mobile clients, location-based context information provides valuable input regarding a service requester's preferences [202]. Here, foundations from the field of location-based search for P2P services as proposed by Kovačević could provide ideas to incorporate such information in the service discovery process [147].

Bibliography

[1] Serge Abiteboul, Peter Buneman, and Dan Suciu. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan Kaufmann Publishers, San Francisco, CA, USA, 1999.

[2] Witold Abramowicz, Konstanty Haniewicz, Monika Kaczmarek, and Dominik Zyskowski. Architecture for Web Services Filtering and Clustering. In *International Conference on Internet and Web Applications and Services (ICIW 2007)*, pages 18–23. IEEE Computer Society, Washington, DC, USA, 2007.

[3] Rama Akkiraju. Semantic Web Services. In Jorge Cardoso, editor, *Semantic Web Services: Theory, Tools, and Applications*, Premier Reference Source, chapter IX, pages 191–216. IGI Global, Hershey, PA, USA, 2007.

[4] Rama Akkiraju and Brahmananda Sapkota, editors. *Semantic Annotations for WSDL and XML Schema – Usage Guide*. W3C Working Group Note, August 2007. `http://www.w3.org/TR/sawsdl-guide/`, access at 2010-01-21.

[5] Rama Akkiraju, Richard Goodwin, Prashant Doshi, and Sascha Roeder. A Method for Semantically Enhancing the Service Discovery Capabilities of UDDI. In *Workshop on Information Integration on the Web (IIWeb-03) at Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*, pages 87–92, 2003.

[6] Rama Akkiraju, Joel Farrell, John Miller, Meenakshi Nagarajan, Marc-Thomas Schmidt, Amit P. Sheth, and Kunal Verma. *Web Service Semantics – WSDL-S*. W3C Member Submission, November 2005. `http://www.w3.org/Submission/WSDL-S/`, access at 2009-09-19.

[7] H. Peter Alesso and Craig F. Smith. *Developing Semantic Web services*. A K Peters, Ltd., Wellesey, MA, USA, 2004.

[8] Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju. *Web Services – Concepts, Architectures and Applications*. Springer, Berlin Heidelberg, 2003.

[9] Alexandre Alves, Assaf Arkin, Sid Askary, Charlton Barreto, Ben Bloch, Francisco Curbera, Mark Ford, Yaron Goland, Alejandro Guízar, Neelakantan Kartha, Canyang Kevin Liu, Rania Khalaf, Dieter König, Mike Marin, Vinkesh Mehta, Satish Thatte, Danny van der Rijn, Prasad Yendluri, and Alex Yiu, editors. *Web Services Business Process Execution Language Version 2.0*. OASIS Standard, April 2007. `http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf`, access at 2010-01-24.

[10] Daniel Austin, Abbie Barbir, Christopher Ferris, and Sharad Garg, editors. *Web Services Architecture Requirements*. W3C Working Group Note, February 2004. `http://www.w3.org/TR/wsa-reqs/`, access at 2009-07-25.

[11] Franz Baader and Werner Nutt. Basic Description Logics. In Franz Baader, Diego Calvanese, Deborah L. McGuiness, Daniele Nardi, and Peter F. Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation and Applications*, chapter 2, pages 47–100. Cambridge University Press, New York, NY, USA, 1. edition, 2003.

[12] Franz Baader, Diego Calvanese, Deborah L. McGuiness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, New York, NY, USA, 1. edition, 2003.

[13] Daniel Bachlechner, Katharina Siorpaes, Dieter Fensel, and Ioan Toma. *Web Service Discovery – A Reality Check*. DERI Technical Report, January 2006. `http://www.sti-innsbruck.at/fileadmin/documents/DERI-TR-2006-01-17.pdf`, access at 2010-01-24.

[14] Ricardo A. Baeza-Yates and Berthier A. Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

[15] Ziv Baida, Jaap Gordijn, and Borys Omelayenko. A Shared Service Terminology for Online Service Provisioning. In *6th International Conference on Electronic Commerce (ICEC 2004)*, volume 60 of *ACM International Conference Proceeding Series*, pages 1–10. ACM, New York, NY, USA, 2004.

[16] Dave Beckett, editor. *RDF/XML Syntax Specification (Revised)*. W3C Recommendation, February 2004. `http://www.w3.org/TR/rdf-syntax-grammar/`, access at 2010-03-06.

[17] Umesh Bellur and Roshan Kulkarni. Improved Matchmaking Algorithm for Semantic Web Services Based on Bipartite Graph Matching. In *2007 IEEE International Conference on Web Services (ICWS 2007)*, pages 86–93. IEEE Computer Society, Washington, DC, USA, 2007.

[18] Umesh Bellur and Harin Vadodaria. On Extending Semantic Matchmaking to Include Preconditions and Effects. In *2008 IEEE International Conference on Web Services (ICWS 2008)*, pages 120–128. IEEE Computer Society, Washington, DC, USA, 2008.

[19] Umesh Bellur and Harin Vadodaria. Web Service Ranking Using Semantic Profile Information. In *IEEE 7th International Conference on Web Services (ICWS 2009)*, pages 872–879. IEEE Computer Society, Washington, DC, USA, 2009.

[20] Umesh Bellur, Harin Vadodaria, and Amit Gupta. Semantic Matchmaking Algorithms. In Witold Bednorz, editor, *Advances in Greedy Algorithms*, chapter 26, pages 481–502. IN-TECH Publications, Vienna, Austria, 2008.

[21] Boualem Benatallah, Mohand-Said Hacid, Alain Léger, Christophe Rey, and Farouk Toumani. On automating Web services discovery. *The VLDB Journal*, 14(1):84–96, 2005.

[22] V. Richard Benjamins, John Davies, Ricardo A. Baeza-Yates, Peter Mika, Hugo Zaragoza, Mark Greaves, Jose Manuél Gómez-Pérez, Jesús Contreras, John Domingue, and Dieter Fensel. Near-Term Prospects for Semantic Technologies. *IEEE Intelligent Systems*, 23(1):76–88, 2008.

[23] Rainer Berbner. *Dienstgüteunterstützung für Service-orientierte Workflows*. Books on Demand GmbH, Norderstedt, Germany, 2008.

[24] Rainer Berbner, Oliver Heckmann, and Ralf Steinmetz. An Architecture for a QoS driven composition of Web Service based Workflows. In *Networking and Electronic Commerce Research Conference (NAEC 2005)*, 2005.

[25] Rainer Berbner, Michael Spahn, Nicolas Repp, Oliver Heckmann, and Ralf Steinmetz. Heuristics for QoS-aware Web Service Composition. In *2006 IEEE International Conference on Web Services (ICWS 2006)*, pages 72–82. IEEE Computer Society, Washington, DC, USA, 2006.

[26] Tim Berners-Lee. Artificial Intelligence and the Semantic Web: AAAI 2006 Keynote. In *The Twenty-First National Conference on Artificial Intelligence (AAAI 2006)*. 2006. `http://www.w3.org/2006/Talks/0718-aaai-tbl/Overview.html`, access at 2009-11-01.

[27] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web: A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities. *Scientific American*, pages 34–43, May 2001.

[28] Abraham Bernstein and Mark Klein. Towards High-Precision Service Retrieval. In *The Semantic Web: First International Semantic Web Conference (ISWC 2002)*, volume 2342 of *Lecture Notes in Computer Science*, pages 84–101. Springer, Berlin Heidelberg, 2002.

[29] Abraham Bernstein, Esther Kaufmann, Christoph Kiefer, and Christoph Bürki. SimPack: A Generic Java Library for Similiarity Measures in Ontologies. Technical report, Department of Informatics, University of Zurich, 2005.

[30] Martin Bichler and Kwei-Jay Lin. Service-Oriented Computing. *Computer*, 39(3):99–101, 2006.

[31] Lianne Bodenstaff, Andreas Wombacher, Manfred Reichert, and Michael C. Jaeger. Analyzing Impact Factors on Composite Services. In *2009 IEEE International Conference on Services Computing (SCC 2009)*, pages 218–226. IEEE Computer Society, Washington, DC, USA, 2009.

[32] David Booth and Canyang Kevin Liu, editors. *Web Service Description Language (WSDL) Version 2.0 Part 0: Primer*. W3C Recommendation, June 2007. `http://www.w3.org/TR/wsdl20-primer/`, access at 2009-07-04.

[33] David Booth, Hugo Haas, Francis McCabe, Eric Newcomer, Michael Champion, Chris Ferris, and David Orchard, editors. *Web Services Architecture Requirements*. W3C Working Group Note, February 2004. `http://www.w3.org/TR/ws-arch/`, access at 2010-02-27.

[34] Alex Borgida and Ronald J. Brachman. Conceptual Modeling with Description Logics. In Franz Baader, Diego Calvanese, Deborah L. McGuiness, Daniele Nardi, and Peter F. Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation and Applications*, chapter 10, pages 359–381. Cambridge University Press, New York, NY, USA, 1. edition, 2003.

[35] François Bourgeois and Jean-Claude Lassalle. An extension of the Munkres algorithm for the assignment problem to rectangular matrices. *Communications of the ACM*, 14(12):802–804, 1971.

[36] Dan Brickley and R.V. Guha, editors. *RDF Vocabulary Description Language 1.0: RDF Schema*. W3C Recommendation, February 2004. `http://www.w3.org/TR/rdf-schema/`, access at 2009-11-01.

[37] Saartje Brockmans, Raphael Volz, Andreas Eberhart, and Peter Löffler. Visual Modeling of OWL DL Ontologies Using UML. In *The Semantic Web: Third International Semantic Web Conference (ISWC 2004)*, volume 3298 of *Lecture Notes in Computer Science*, pages 198–213. Springer, Berlin Heidelberg, 2004.

[38] Jeen Broekstra and Arjohn Kampman. An RDF Query and Transformation Language. In Steffen Staab and Heiner Stuckenschmidt, editors, *Semantic Web and Peer-to-Peer: Decentralized Management and Exchange of Knowledge and Information*, chapter 1, pages 23–39. Springer, Berlin Heidelberg, 2006.

[39] François Bry, Tim Furche, Benedikt Linse, Alexander Pohl, Antonius Weinzierl, and Olga Yestekhina. Four Lessons in Versatility or How Query Languages Adapt to the Web. In François Bry and Jan Maluszynski, editors, *Semantic Techniques for the Web, The REWERSE Perspective*, volume 5500 of *Lecture Notes in Computer Science*, chapter 2, pages 50–160. Springer, Berlin Heidelberg, 2009.

[40] Alexander Budanitsky and Graeme Hirst. Evaluating WordNet-based Measures of Lexical Semantic Relatedness. *Computational Linguistics*, 32(1):13–47, 2006.

[41] Rainer Burkard, Mauro Dell'Amico, and Silvano Martello. *Assignment Problems*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2009.

[42] Christoph Bussler. Semantic Web Services. In Dieter Fensel, Mick Kerrigan, and Michal Zaremba, editors, *Implementing Semantic Web Services. The SESA Framework*, chapter 2, pages 27–41. Springer, Berlin Heidelberg, 2008.

[43] Christoph Bussler, Dieter Fensel, and Alexander Mädche. A Conceptual Architecture for Semantic Web Enabled Web Services. *SIGMOD Record*, 31(4):24–29, 2002.

[44] Liliana Cabral, John Domingue, Enrico Motta, Terry R. Payne, and Farshad Hakimpour. Approaches to Semantic Web Services: an Overview and Comparisons. In *The Semantic Web: Research and Applications, First European Semantic Web Symposium (ESWS 2004)*, volume 3053 of *Lecture Notes in Computer Science*, pages 225–239. Springer, Berlin Heidelberg, 2004.

[45] Jorge Cardoso. Discovering Semantic Web Services with and without a Common Ontology Commitment. In *IEEE Services Computing Workshops (SCW '06), Third International Semantic and Dynamic Web Processes Workshop (SDWP 2006)*, pages 183–190. IEEE Computer Society, Washington, DC, USA, 2006.

[46] Jorge Cardoso. The Syntactic and the Semantic Web. In Jorge Cardoso, editor, *Semantic Web Services: Theory, Tools, and Applications*, Premier Reference Source, chapter I, pages 1–23. IGI Global, Hershey, PA, USA, 2007.

[47] Jorge Cardoso and Amit P. Sheth. Semantic E-Workflow Composition. *Journal of Intelligent Information Systems*, 21(3):191–225, 2003.

[48] Jorge Cardoso, Amit P. Sheth, John A. Miller, Jonathan Arnold, and Krys Kochut. Quality of Service for Workflows and Web Service Processes. *Journal of Web Semantics*, 1(3):281–308, 2004.

[49] Jorge Cardoso, Konrad Voigt, and Matthias Winkler. Service Engineering for the Internet of Services. In *10th International Conference on Enterprise Information Systems (ICEIS 2008)*, volume 19 of *Lecture Notes in Business Information Processing*, pages 15–27. Springer, Berlin Heidelberg, 2008.

[50] Jorge Cardoso, Matthias Winkler, and Konrad Voigt. A Service Description Language for the Internet of Services. In *First International Symposium on Services Science (ISSS'09)*, volume 5 of *Leipziger Beiträge zur Wirtschaftsinformatik*, pages 229–240. Logos Verlag, Berlin, 2009.

[51] Jeremy J. Carroll, Ian Dickinson, Chris Dollin, Dave Reynolds, Andy Seaborne, and Kevin Wilkinson. Jena: Implementing the Semantic Web Recommendations. In *13th International World Wide Web Conference – Alternate Track Papers & Posters (WWW 2004)*, pages 74–83. ACM, New York, NY, USA, 2004.

[52] B. Chandrasekaran, John R. Josephson, and V. Richard Benjamins. What Are Ontologies, and Why Do We Need Them? *IEEE Intelligent Systems*, 14(1):20–26, 1999.

[53] Chih-Chung Chang and Chih-Jen Lin. *LIBSVM: A Library for Support Vector Machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`, access at 2010-01-24.

[54] Pierre Châtel. *Service Registries Study*. Thales Study, June 2006. `http://www.chatelp.org/work/LUCAS_registry_study.pdf`, last access at 2010-04-13.

[55] Mao Chen. A Greedy Algorithm with Forward-Looking Strategy. In Witold Bednorz, editor, *Advances in Greedy Algorithms*, chapter 1, pages 1–16. IN-TECH Publications, Vienna, Austria, 2008.

[56] Roberto Chinnici, Hugo Haas, Amelia A. Lewis, Jean-Jacques Moreau, David Orchard, and Sanjiva Weerawarana, editors. *Web Services Description Language (WSDL) Version 2.0 Part 2: Adjuncts*. W3C Recommendation, June 2007. `http://www.w3.org/TR/wsdl20-adjuncts`, access at 2010-01-24.

[57] Roberto Chinnici, Jean-Jacques Moreau, Arthur Ryman, and Sanjiva Weerawarana, editors. *Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language*. W3C Recommendation, June 2007. `http://www.w3.org/TR/wsdl20/`, access at 2010-01-24.

[58] Joseph M. Chiusano and Farrukh Najmi. *Registering Web Services in an ebXML Registry, Version 1.0*. OASIS Technical Note, March 2003. `http://www.oasis-open.org/committees/download.php/11907/regrep-webservices-tn-10.pdf`, access at 2010-01-24.

[59] Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. *Web Services Description Language (WSDL) 1.1*. W3C Note, March 2001. `http://www.w3.org/TR/wsdl.html`, access at 2010-01-24.

[60] Emilia Cimpian and Michal Zaremba, editors. *Web Service Execution Environment (WSMX)*. W3C Member Submission, June 2005. `http://www.w3.org/Submission/WSMX/`, access at 2010-04-03.

[61] Emilia Cimpian, Harald Meyer, Dumitru Roman, Adina Sirbu, Nathalie Steinmetz, Steffen Staab, and Ioan Toma. Ontologies and Matchmaking. In Dominik Kuropka, Pter Tröger, Steffen Staab, and Mathias Weske, editors, *Semantic Service Provisioning*, pages 19–54. Springer, Berlin Heidelberg, 2008.

[62] Luc Clement, Andrew Hately, Claus von Riegen, and Tony Rogers, editors. *UDDI Version 3.0.2*. UDDI Spec Technical Committee Draft, October 2004. `http://uddi.org/pubs/uddi_v3.htm`, access at 2010-01-24.

[63] John Colgrave and Karsten Januszewski. *Using WSDL in a UDDI Registry, Version 2.0.2*. OASIS Technical Note, June 2004. `http://www.oasis-open.org/committees/uddi-spec/doc/tn/uddi-spec-tc-tn-wsdl-v2.htm`, access at 2010-01-24.

[64] John Colgrave, Rama Akkiraju, and Richard Goodwin. External Matching in UDDI. In *IEEE International Conference on Web Services (ICWS'04)*, pages 226–233. IEEE Computer Society, Washington, DC, USA, 2004.

[65] Claudia d'Amato, Steffen Staab, Nicola Fanizzi, and Floriana Esposito. Efficient Discovery of Services Specified in Description Logics Languages. In *First International Joint Workshop SMR^2 2007 on Service Matchmaking and Resource Retrieval in the Semantic Web at the 6th International Semantic Web Conference (ISWC 2007)*, volume 243 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.

[66] Doug Davis, Anish Karmarkar, Gilbert Pilz, Steve Winkler, and Ümit Yalçinalp, editors. *Web Services Reliable Messaging (WS-ReliableMessaging) Version 1.2*. OASIS Standard, February 2009. `http://docs.oasis-open.org/ws-rx/wsrm/200702/wsrm-1.2-spec-os.html`, access at 2009-11-14.

[67] Mike Dean and Guus Schreiber, editors. *OWL Web Ontology Language Reference*. W3C Recommendation, February 2004.

[68] Tommaso Di Noia, Eugenio Di Sciascio, and Francesco M. Donini. Semantic Matchmaking as Non-Monotonic Reasoning: A Description Logic Approach. *Journal of Artificial Intelligence Research*, 29 (1):269–307, 2007.

[69] Tharam S. Dillon, Chen Wu, and Elizabeth Chang. Reference Architectural Styles for Service-Oriented Computing. In *IFIP International Conference on Network and Parallel Computing (NPC 2007)*, volume 4672 of *Lecture Notes in Computer Science*, pages 543–555. Springer, Berlin Heidelberg, 2007.

[70] Asuman Dogac, Yildiray Kabak, and Gokce B. Laleci. Enriching ebXML Registries with OWL Ontologies for Efficient Service Discovery. In *14th International Workshop on Research Issues in Data Engineering: Web Services for E-Commerce and E-Government Applications (RIDE 2004)*, pages 69–76, 2004.

[71] Asuman Dogac, Yildiray Kabak, Gokce Laleci, Carl Mattocks, Farrukh Najmi, and Jeff Pollock. Enhancing ebXML Registries to Make them OWL Aware. *Distributed and Parallel Databases*, 18(1): 9–36, 2005.

[72] John Domingue, Dieter Fensel, and Rafael González-Cabero. SOA4All, Enabling the SOA Revolution on a World Wide Scale. In *2008 IEEE International Conference on Semantic Computing (ICSC '08)*, pages 530–537. IEEE Computer Society, Washington, DC, USA, 2008.

[73] Xin Dong, Alon Halevy, Jayant Madhavan, Ema Nemes, and Jun Zhang. Simlarity Search for Web Services. In *Thirtieth international conference on Very large data bases (VLDB '04)*, pages 372–383. Morgan Kaufmann Publishers, San Francisco, CA, USA, 2004.

[74] Julian Eckert. *Cross-organizational Service-based Workflows – Solution Strategies for Quality of Service Optimization*. TU Darmstadt, 2009. Dissertation.

[75] Marc Ehrig and York Sure. Ontology Mapping – An Integrated Approach. In *The Semantic Web: Research and Applications, First European Semantic Web Symposium (ESWS 2004)*, volume 3053 of *Lecture Notes in Computer Science*, pages 76–91. Springer, Berlin Heidelberg, 2004.

[76] Thomas Erl. *Service-Oriented Architecture: Concepts, Technology, and Design*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005.

[77] Andreas Faatz. *Ein Verfahren zur Anreicherung fachgebietsspezifischer Ontologien durch Begriffsvorschläge*. TU Darmstadt, 2004. Dissertation.

[78] David C. Fallside and Priscilla Walmsley, editors. *XML Schema Part 0: Primer Second Edition*. W3C Recommendation, October 2004. http://www.w3.org/TR/xmlschema-0/, access at 2010-03-04.

[79] Joel Farrell and Holger Lausen, editors. *Semantic Annotations for WSDL and XML Schema*. W3C Recommendation, August 2007. http://www.w3.org/TR/sawsdl/, access at 2010-02-09.

[80] Max Feingold and Ram Jeyaraman, editors. *Web Services Coordination Version (WS-Coordination) 1.2*. OASIS Standard, February 2009. http://docs.oasis-open.org/ws-tx/wstx-wscoor-1.2-spec-os.doc, access at 2010-02-09.

[81] Dieter Fensel, Holger Lausen, Axel Polleres, Joe de Bruijn, Michael Stollberg, Dumitru Roman, and John Domingue. *Enabling Semantic Web Services: The Web Service Modeling Ontology*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[82] Alberto Fernández, Axel Polleres, and Sascha Ossowski. Towards Fine-grained Service Matchmaking by Using Concept Similarity. In *First International Joint Workshop SMR^2 2007 on Service Matchmaking and Resource Retrieval in the Semantic Web at the 6th International Semantic Web Conference (ISWC 2007)*, volume 243 of *CEUR Workshop Proceedings*, pages 31–46. CEUR-WS.org, 2007.

[83] W. Nelson Francis and Henry Kucera, editors. *BROWN CORPUS MANUAL, revised and amplified edition*. Department of Linguistics, Brown University, 1979. http://khnt.aksis.uib.no/icame/manuals/brown/, access at 2010-03-09.

[84] Tom Freund and Mark Little, editors. *Web Services Business Activity (WS-BusinessActivity) Version 1.2*. OASIS Standard, February 2009. http://docs.oasis-open.org/ws-tx/wstx-wsba-1.2-spec-os.doc, access at 2010-02-09.

[85] Sally Fuger, Farruk Najmi, and Nikola Stojanovic, editors. *ebXML Registry Information Model Version 3.0*. OASIS Standard, May 2005. http://docs.oasis-open.org/regrep/v3.0/specs/regrep-rim-3.0-os.pdf, access at 2010-02-09.

[86] Sally Fuger, Farruk Najmi, and Nikola Stojanovic, editors. *ebXML Registry Services and Protocols Version 3.0*. OASIS Standard, May 2005. http://docs.oasis-open.org/regrep/v3.0/specs/regrep-rs-3.0-os.pdf, access at 2010-02-09.

[87] Karthik Gomadam, Kunal Verma, Amit P. Sheth, and Ke Li. Keywords, Port Types and Semantics: A Journey in the Land of Web Service Discovery. In Jorge Cardoso and Amit P. Sheth, editors, *Semantic Web Services, Processes and Applications*, Semantic Web and Beyond: Computing for Human Experience, chapter 4, pages 89–105. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[88] Karl D. Gottschalk, Stephen Graham, Heather Kreger, and James Snell. Introduction to Web services architecture. *IBM Systems Journal*, 41(2):170–177, 2002.

[89] Frederick J. Gravetter and Larry B. Wallnau. *Statistics for the Behavioral Sciences*. Wadsworth Cengage Learning, Belmont, CA, USA, 7th edition, 2006.

[90] Stephan Grimm. Discovery. In Rudi Studer, Stephan Grimm, and Andreas Abecker, editors, *Semantic Web Services. Concepts, Technologies, and Applications*, chapter 8, pages 211–244. Springer, Berlin Heidelberg, 2007.

[91] Stephan Grimm and Pascal Hitzler. Semantic Matchmaking of Web Resources with Local Closed-World Reasoning. *International Journal of Electronic Commerce*, 12(2):89–126, 2008.

[92] Stephan Grimm, Boris Motik, and Chris Preist. Matching Semantic Service Descriptions with Local Closed-World Reasoning. In *The Semantic Web: Research and Applications, 3rd European Semantic Web Conference (ESWC 2006)*, volume 4011 of *Lecture Notes in Computer Science*, pages 575–589. Springer, Berlin Heidelberg, 2006.

[93] David A. Grossman and Ophir Frieder. *Information Retrieval: Algorithms and Heuristics*. The Kluwer International Series of Information Retrieval. Springer, Dordrecht, The Netherlands, second edition, 2004.

[94] Tom Gruber. A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.

[95] Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, Henrik Frystyk Nielsen, Anish Karmarkar, and Yves Lafon, editors. *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)*. W3C Recommendation, April 2007. `http://www.w3.org/TR/soap12-part1/`, access at 2010-02-09.

[96] Martin Gudgin, Marc Hadley, Noah Mendelsohn, Jean-Jacques Moreau, Henrik Frystyk Nielsen, Anish Karmarkar, and Yves Lafon, editors. *SOAP Version 1.2 Part 2: Adjuncts (Second Edition)*. W3C Recommendation, April 2007. `http://www.w3.org/TR/soap12-part2/`, access at 2010-02-09.

[97] Ruiqiang Guo, Dehua Chen, and Jiajin Le. Matching Semantic Web Services across Heterogeneous Ontologies. In *Fifth International Conference on Computer and Information Technology (CIT 2005)*, pages 264–268. IEEE Computer Society, Washington, DC, USA, 2005.

[98] Peter Haase, Sudhir Agarwal, and York Sure. Service-Oriented Semantic Peer-to-Peer Systems. In *International Workshop on Intelligent Networked and Mobile Systems at The Fifth International Conference on Web Information Systems Engineering (WISE 2004)*, volume 3307 of *Lecture Notes in Computer Science*, pages 46–57. Springer, Berlin Heidelberg, 2004.

[99] Peter Haase, Frank van Harmelen, Zhisheng Huang, Heiner Stuckenschmidt, and York Sure. A Framework for Handling Inconsistency in Changing Ontologies. In *The Semantic Web: 4th International Semantic Web Conference (ISWC 2005)*, volume 3729 of *Lecture Notes in Computer Science*, pages 353–367. Springer, Berlin Heidelberg, 2005.

[100] Marc Hadley. *Web Application Description Language*. W3C Member Submission, August 2009. `http://www.w3.org/Submission/wadl/`, access at 2010-03-30.

[101] Armin Haller, Emilia Cimpian, Adrian Mocan, Eyal Oren, and Christoph Bussler. WSMX – A Semantic Service-Oriented Architecture. In *2005 IEEE International Conference on Web Services (ICWS 2005)*, pages 321–328. IEEE Computer Society, Washington, DC, USA, 2005.

[102] Martin Hepp. Products and Services Ontologies: A Methodology for Deriving OWL Ontologies from Industrial Categorization Standards. *International Journal on Semantic Web and Information Systems*, 2(1):72–99, 2006.

[103] Martin Hepp. Possible Ontologies: How Reality Constrains the Development of Relevant Ontologies. *IEEE Internet Computing*, 11(1):90–96, 2007.

[104] Martin Hepp. An Ontology for Describing Products and Services Offers on the Web. In *16th International Conference on Knowledge Engineering and Knowledge Management (EKAW2008)*, volume 5268 of *Lecture Notes on Computer Science*, pages 332–347. Springer, Berlin Heidelberg, 2008.

[105] Martin Hepp, Frank Leymann, John Domingue, Alexander Wahler, and Dieter Fensel. Semantic Business Process Management: A Vision Towards Using Semantic Web Services for Business Process Management. In *IEEE International Conference on e-Business Engineering (ICEBE 2005)*, pages 535–540. IEEE Computer Society, Washington, DC, USA, 2005.

[106] Reinhold Herzog, Holger Lausen, Dumitru Roman, and Peter Zugmann, editors. *D10 v0.1 WSMO Registry*. WSMO Working Draft, April 2004. `http://www.wsmo.org/2004/d10/v0.1/20040426/`, access at 2010-04-03.

[107] Andreas Heß, Eddie Johnston, and Nicholas Kushmerick. ASSAM: A Tool for Semi-automatically Annotating Semantic Web Services. In *The Semantic Web: Third International Semantic Web Conference (ISWC 2004)*, volume 3298 of *Lecture Notes in Computer Science*, pages 320–334. Springer, Berlin Heidelberg, 2004.

[108] Pascal Hitzler, Markus Krötzsch, Bijan Parsia, Peter F. Patel-Schneider, and Sebastian Rudolph, editors. *OWL 2 Web Ontology Language: Primer*. W3C Recommendation, October 2009.

[109] Ian Horrocks. DAML+OIL: A Reason-able Web Ontology Language. In *Advances in Database Technology: 8th International Conference on Extending Database Technology (EDBT 2002)*, volume 2287 of *Lecture Notes in Computer Science*, pages 2–13. Springer, Berlin Heidelberg, 2002.

[110] Ian Horrocks, Peter F. Patel-Schneider, and Frank van Harmelen. From SHIQ and RDF to OWL: the making of a Web Ontology Language. *Journal of Web Semantics*, 1(1):7–26, 2003.

[111] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*. W3C Member Submission, May 2004. `http://www.w3.org/Submission/SWRL/`, access at 2009-10-09.

[112] Ian Horrocks, Peter F. Patel-Schneider, Deborah L. McGuiness, and Christopher A. Welty. OWL: a Description Logic Based Ontology for the Semantic Web. In Franz Baader, Diego Calvanese, Deborah L. McGuiness, Daniele Nardi, and Peter F. Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation and Applications*, chapter 14, pages 458–486. Cambridge University Press, New York, NY, USA, 2nd edition, 2007.

[113] Kashif Iqbal, Marco Luca Sbodio, Vassilios Peristeras, and Giovanni Giuliani. Semantic Service Discovery using SAWSDL and SPARQL. In *Fourth International Conference on Semantics, Knowledge and Grid (SKG 2008)*, pages 205–212. IEEE Computer Society, Washington, DC, USA, 2008.

[114] Michael C. Jaeger and Stefan Tang. Ranked Matching for Service Descriptions using DAML-S. In *Open INTEROP Workshop on Workshop on Enterprise Modelling and Ontologies for Interoperability (EMOI – INTEROP 2004) in connection with The 16th Conference on Advanced Information Systems Engineering (CAiSE 2004), Vol. 3*, pages 217–228, 2004.

[115] Michael C. Jaeger, Gregor Rojec-Goldmann, Christoph Liebetruth, Gero Mühl, and Kurt Geihs. Ranked Matching for Service Descriptions Using OWL-S. In *14. ITG/GI-Fachtagung Kommunikation in Verteilten Systemen (KiVS 2005)*, Informatik Aktuell, pages 91–102. Springer, Berlin Heidelberg, 2005.

[116] Nicolai M. Josuttis. *SOA in Practice: The Art of Distributed System Design*. O'Reilly Media Inc., Sebastopol, CA, USA, 2007.

[117] Vipul Kashyap and Amit P. Sheth. Semantics-Based Information Brokering. In *Third International Conference on Information and Knowledge Management (CIKM'94)*, pages 363–370. ACM, New York, NY, USA, 1994.

[118] Vipul Kashyap, Christoph Bussler, and Matthew Moran. *The Semantic Web, Semantics for Data and Services on the Web*. Springer-Verlag, Berlin, Heidelberg, 2008.

[119] Frank Kaufer and Matthias Klusch. WSMO-MX: A Logic Programming Based Hybrid Service Matchmaker. In *Fourth IEEE European Conference on Web Services (ECOWS 2006)*, pages 161–170. IEEE Computer Society, Washington, DC, USA, 2006.

[120] Frank Kaufer and Matthias Klusch. Performance of Hybrid WSML Service Matching with WSMO-MX: Preliminary Results. In *First International Joint Workshop SMR^2 2007 on Service Matchmaking and Resource Retrieval in the Semantic Web at the 6th International Semantic Web Conference (ISWC 2007)*, volume 243 of *CEUR Workshop Proceedings*, pages 63–77. CEUR-WS.org, 2007.

[121] Peter Kennedy. *A Guide to Econometrics*. The MIT Press, Cambridge, MA, USA, 5 edition, 2003.

[122] Mahboob Alam Khalid, Benedikt Fries, Patrick Kapahnke, and Matthias Klusch. *SAWSDL-TC 1*. DFKI Saarbrücken, Germany, 2008. `http://projects.semwebcentral.org/projects/sawsdl-tc/`, last access on 2010-02-015.

[123] Christoph Kiefer and Abraham Bernstein. The Creation and Evaluation of iSPARQL Strategies for Matchmaking. In *The Semantic Web: Research and Applications, 5th European Semantic Web Conference (ESWC 2008)*, volume 5021 of *Lecture Notes in Computer Science*, pages 463–477. Springer, Berlin Heidelberg, 2008.

[124] Christoph Kiefer, Abraham Bernstein, Hong Joo Lee, Mark Klein, and Markus Stocker. Semantic Process Retrieval with iSPARQL. In *The Semantic Web: Research and Applications, 4th European Semantic Web Conference (ESWC 2007)*, volume 4519 of *Lecture Notes in Computer Science*, pages 609–623. Springer, Berlin Heidelberg, 2007.

[125] Christoph Kiefer, Abraham Bernstein, and Markus Stocker. The Fundamentals of iSPARQL – A Virtual Triple Approach For Similarity-Based Semantic Web Tasks. In *The Semantic Web: 6th International and 2nd Asian Semantic Web Conference (ISWC 2007 + ASWC 2007)*, volume 4825 of *Lecture Notes in Computer Science*, pages 295–308. Springer, Berlin Heidelberg, 2007.

[126] Kjetil Kjernsmo and Alexandre Passant, editors. *SPARQL New Features and Rationale*. W3C Working Draft, July 2009. `http://www.w3.org/TR/sparql-features/`, access at 2010-04-15.

[127] Mark Klein and Abraham Bernstein. Toward High-Precision Service Retrieval. *IEEE Internet Computing*, 8(1):30–36, 2004.

[128] Michael Klein and Birgitta König-Ries. Coupled Signature and Specification Matching for Automatic Service Binding. In *European Conference on Web Services (ECOWS 2004)*, volume 3250 of *Lecture Notes in Computer Science*, pages 183–197. Springer, Berlin Heidelberg, 2004.

[129] Matthias Klusch. Semantic Web Service Description. In Michael Schumacher, Heikki Helin, and Heiko Schuldt, editors, *CASCOM: Intelligent Service Coordination in the Semantic Web*, Whitestein Series in Software Agent Technologies and Autonomic Computing, chapter 3, pages 31–57. Birkhäuser Verlag, Basel, Switzerland, 2008.

[130] Matthias Klusch. Semantic Web Service Coordination. In Michael Schumacher, Heikki Helin, and Heiko Schuldt, editors, *CASCOM: Intelligent Service Coordination in the Semantic Web*, Whitestein Series in Software Agent Technologies and Autonomic Computing, chapter 4, pages 59–104. Birkhäuser Verlag, Basel, Switzerland, 2008.

[131] Matthias Klusch and Benedikt Fries. Hybrid OWL-S Service Retrieval with OWLS-MX: Benefits and Pitfalls. In *First International Joint Workshop SMR^2 2007 on Service Matchmaking and Resource Retrieval in the Semantic Web at the 6th International Semantic Web Conference (ISWC 2007)*, volume 243 of *CEUR Workshop Proceedings*, pages 47–61. CEUR-WS.org, 2007.

[132] Matthias Klusch and Patrick Kapahnke. Semantic Web Service Selection with SAWSDL-MX. In *Second International Workshop SMR^2 2008 on Service Matchmaking and Resource Retrieval in the Semantic Web at the 7th International Semantic Web Conference (ISWC 2008)*, volume 416 of *CEUR Workshop Proceedings*, pages 3–18. CEUR-WS.org, 2008.

[133] Matthias Klusch and Patrick Kapahnke. OWLS-MX3: An Adaptive Hybrid Semantic Service Matchmaker for OWL-S. In *Third International Workshop SMR^2 2009 on Service Matchmaking and Resource Retrieval in the Semantic Web at the 8th International Semantic Web Conference (ISWC 2009)*, volume 525 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009.

[134] Matthias Klusch and Frank Kaufer. WSMO-MX: A hybrid Semantic Web service matchmaker. *Web Intelligence and Agent Systems*, 7(1):23–42, 2009.

[135] Matthias Klusch and Xiguo Zhing. Deployed Semantic Services for the Common User of the Web: A Reality Check. In *Second IEEE International Conference on Semantic Computing (ICSC 2008)*, pages 347–353. IEEE Computer Society, Washington, DC, USA, 2008.

[136] Matthias Klusch, Benedikt Fries, and Katia Sycara. Automated Semantic Web Service Discovery with OWLS-MX. In *5th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2006)*, pages 915–922. ACM, New York, NY, USA, 2006.

[137] Matthias Klusch, Patrick Kapahnke, and Benedikt Fries. Hybrid Semantic Web Service Retrieval: A Case Study with OWLS-MX. In *2nd IEEE International Conference on Semantic Computing (ICSC 2008)*, pages 323–330. IEEE Computer Society, Washington, DC, USA, 2008.

[138] Matthias Klusch, Patrick Kapahnke, and Frank Kaufer. Evaluation of WSML Service Retrieval with WSMO-MX. In *2008 IEEE International Conference on Web Services (ICWS 2008)*, pages 401–408. IEEE Computer Society, Washington, DC, USA, 2008.

[139] Matthias Klusch, Benedikt Fries, and Katia P. Sycara. OWLS-MX: A hybrid Semantic Web service matchmaker for OWL-S services. *Journal of Web Semantics*, 7(2):121–133, 2009.

[140] Matthias Klusch, Patrick Kapahnke, and Ingo Zinnikus. Hybrid Adaptive Web Service Selection with SAWSDL-MX and WSDL-Analyzer. In *The Semantic Web: Research and Applications, 6th European Semantic Web Conference (ESWC 2009)*, volume 5554 of *Lecture Notes in Computer Science*, pages 550–564. Springer, Berlin Heidelberg, 2009.

[141] Matthias Klusch, Patrick Kapahnke, and Ingo Zinnikus. SAWSDL-MX2: A Machine-Learning Approach for Integrating Semantic Web Service Matchmaking Variants. In *IEEE 7th International Conference on Web Services (ICWS 2009)*, pages 335–342. IEEE Computer Society, Washington, DC, USA, 2009.

[142] Matthias Klusch, Alain Leger, David Martin, Massimo Paolucci, Abraham Bernstein, and Ulrich Küster. 3rd International Semantic Service Selection Contest – Retrieval Performance Evaluation of Matchmakers for Semantic Web Services (S3 Contest). In *Third International Workshop SMR^2 2009 on Service Matchmaking and Resource Retrieval in the Semantic Web at the 8th International Semantic Web Conference (ISWC 2009)*, 2009.

[143] Jacek Kopecký, editor. *Web Services Description Language (WSDL) Version 2.0: RDF Mapping*. W3C Working Group Note, 2007. `http://www.w3.org/TR/wsdl20-rdf/`, last access on 2010-02-15.

[144] Jacek Kopecký, Tomas Vitvar, Carine Bournez, and Joel Farrell. SAWSDL: Semantic Annotations for WSDL and XML Schema. *IEEE Internet Computing*, 11(6):60–67, 2007.

[145] Dimitrios Kourtesis and Iraklis Paraskakis. Combining SAWSDL, OWL-DL and UDDI for Semantically Enhanced Web Service Discovery. In *The Semantic Web: Research and Applications, 5th European Semantic Web Conference (ESWC 2008)*, volume 5021 of *Lecture Notes in Computer Science*, pages 614–628. Springer, Berlin Heidelberg, 2008.

[146] Dimitrios Kourtesis and Iraklis Paraskakis. Web Service Discovery in the FUSION Semantic Registry. In *11th International Conference on Business Information Systems (BIS 2008)*, volume 7 of *Lecture Notes in Business Information Processing*, pages 285–296. Springer, Berlin Heidelberg, 2008.

[147] Aleksandra Kovačević. *Peer-to-Peer Location-based Search: Engineering a Novel Peer-to-Peer Overlay Network*. TU Darmstadt, 2009. Dissertation.

[148] Dirk Krafzig, Karl Banke, and Dirk Slama. *Enterprise SOA : Service-Oriented Architecture Best Practices*. The Coad Series. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004.

[149] Reto Krummenacher, Martin Hepp, Axel Polleres, Christoph Bussler, and Dieter Fensel. WWW or What Is Wrong with Web Services. In *Third European Conference on Web Services (ECOWS 2005)*, pages 235–243. IEEE Computer Society, Washington, DC, USA, 2005.

[150] Harold William Kuhn. The Hungarian method for the assignment problem. *Naval Research Logistic Quarterly*, 2:83–97, 1955.

[151] Harold William Kuhn. Variants of the Hungarian method for assignment problems. *Naval Research Logistic Quarterly*, 3:253–258, 1956.

[152] Ulrich Küster and Birgitta König-Ries. Evaluating Semantic Web Service Matchmaking Effectiveness Based on Graded Relevance. In *Second International Workshop SMR^2 2008 on Service Matchmaking and Resource Retrieval in the Semantic Web at the 7th International Semantic Web Conference (ISWC 2008)*, volume 416 of *CEUR Workshop Proceedings*, pages 35–50. CEUR-WS.org, 2008.

[153] Ulrich Küster and Birgitta König-Ries. Towards Standard Test Collections for the Empirical Evaluation of Semantic Web Service Approaches. *International Journal of Semantic Computing*, 2(3): 381–402, 2008.

[154] Ulrich Küster and Birgitta König-Ries. Relevance Judgments for Web Services Retrieval – A Methodology and Test Collection for SWS Discovery Evaluation. In *Seventh IEEE European Conference on Web Services (ECOWS 2009)*, pages 17–26. IEEE Computer Society, Washington, DC, USA, 2009.

[155] Ulrich Küster, Birgitta König-Ries, Mirco Stern, and Michael Klein. DIANE: an integrated approach to automated service discovery, matchmaking and composition. In *16th International Conference on World Wide Web (WWW 2007)*, pages 1033–1042. ACM, 2007.

[156] Ulrich Küster, Birgitta König-Ries, Charles J. Petrie, and Matthias Klusch. On the Evaluation of Semantic Web Service Frameworks. *International Journal on Semantic Web and Information Systems*, 4(4):31–55, 2008.

[157] Ulrich Küster, Holger Lausen, and Birgitta König-Ries. Evaluation of Semantic Service Discovery – A Survey and Directions for Future Research. In *2nd Workshop on Emerging Web Services Technology (WEWST 2007) at the Fifth European Conference on Web Services (ECOWS 2007)*, volume 313 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

[158] Lee W. Lacy. *OWL: Representing Information Using the Web Ontology Language*. Trafford Publishing, Victoria, BC, Canada, 2005.

[159] Steffen Lamparter and Anupriya Ankolekar. Automated Selection of Configurable Web Services. In *eOrganisation: Service-, Prozess-, Market-Engineering: 8. Internationale Tagung Wirtschaftsinformatik (WI 2007) – Band 1*, pages 441–458. Universitätsverlag Karlsruhe, Germany, 2007.

[160] Holger Lausen. Discovery. In Dieter Fensel, Mick Kerrigan, and Michal Zaremba, editors, *Implementing Semantic Web Services. The SESA Framework*, chapter 8, pages 167–191. Springer, Berlin Heidelberg, 2008.

[161] Holger Lausen and Nathalie Steinmetz. Survey of Current Means to Discover Web Services. Technical report, STI Innsbruck, Austria, August 2008. `http://www.deri.at/fileadmin/documents/technical_report/TR-STI-2008-08-08.pdf`, access at 2010-02-15.

[162] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady*, 10(8):707–710, 1966.

[163] Frank Leymann, Dieter Roller, and Marc-Thomas Schmidt. Web services and business process management. *IBM Systems Journal*, 41(2):198–211, 2002.

[164] Lei Li and Ian Horrocks. A software framework for matchmaking based on semantic web technology. In *12th International World Wide Web Conference (WWW 2003)*, pages 331–339. ACM, New York, NY, USA, 2003.

[165] Lei Li and Ian Horrocks. A Software Framework for Matchmaking Based on Semantic Web Technology. *International Journal of Electronic Commerce*, 8(4):39–60, 2004.

[166] Yuhua Li, Zuhair Bandar, and David McLean. An Approach for Measuring Semantic Similarity between Words Using Multiple Information Sources. *IEEE Transactions on Knowledge and Data Engineering*, 15(4):871–882, 2003.

[167] Dekang Lin. An Information-Theoretic Definition of Similarity. In *Fifteenth International Conference on Machine Learning (ICML 1998)*, pages 296–304. Morgan Kaufmann Publishers, San Francisco, CA, USA, 1998.

[168] Mark Little and Andrew Wilkinson, editors. *OASIS Web Services Atomic Transaction Version 1.2*. OASIS Standard, February 2009. http://docs.oasis-open.org/ws-tx/wstx-wsat-1.2-spec-os.doc, access at 2010-02-15.

[169] Chuanchang Liu, Yong Peng, and Junliang Chen. Web Services Description Ontology-Based Service Discovery Model. In *2006 IEEE / WIC / ACM International Conference on Web Intelligence (WI 2006)*, pages 633–636. IEEE Computer Society, Washington, DC, USA, 2006.

[170] Jim Luo, Bruce Montrose, Anya Kim, Amitabh Khashnobish, and Myong Kang. Adding OWL-S Support to the Existing UDDI Infrastructure. In *2006 IEEE International Conference on Web Services (ICWS 2006)*, pages 153–162. IEEE Computer Society, Washington, DC, USA, 2006.

[171] C. Matthew MacKenzie, Ken Laskey, Francis McCabe, Peter F Brown, and Rebekah Metz, editors. *Reference Model for Service Oriented Architecture 1.0*. OASIS Standard, August 2006. http://docs.oasis-open.org/soa-rm/v1.0/soa-rm.pdf access at 2010-02-15.

[172] Alexander Mädche and Steffen Staab. Ontology Learning for the Semantic Web. *IEEE Intelligent Systems*, 16(2):72–79, 2001.

[173] Alexander Mädche, Steffen Staab, Nenad Stojanovic, Rudi Studer, and York Sure. SEmantic portAL: The SEAL Approach. In Dieter Fensel, James A. Hendler, Henry Lieberman, and Wolfgang Wahlster, editors, *Spinning the Semantic Web: Bringing the World Wide Web to Its Full Potential*, pages 317–359. The MIT Press, Cambridge, MA, USA, 2003.

[174] Maria Maleshkova. Acquisition and Management of Semantic Web Service Descriptions. In *Ph.D. Symposium at the 5th European Semantic Web Conference (ESWC 2008)*, volume 358 of *CEUR Workshop Proceedings*, pages 41–45. CEUR-WS.org, 2008.

[175] Daniel J. Mandell and Sheila A. McIlraith. Adapting BPEL4WS for the Semantic Web: The Bottom-Up Approach to Web Service Interoperation. In *The Semantic Web: Fourth International Semantic Web Conference (ISWC 2003)*, volume 2870 of *Lecture Notes in Computer Science*, pages 227–241. Springer, Berlin Heidelberg, 2003.

[176] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, New York, NY, USA, 2008.

[177] Frank Manola and Eric Miller, editors. *RDF Primer*. W3C Recommendation, February 2004. http://www.w3.org/TR/rdf-primer/, access at 2009-11-01.

[178] David Martin, Mark Burstein, Drew McDermott, Sheila A. McIlraith, Massimo Paolucci, Katia P. Sycara, Deborah L. McGuinness, Evren Sirin, and Naveen Srinivasan. Bringing Semantics to Web Services with OWL-S. *World Wide Web*, 10(3):243–277, 2007.

[179] David Martin, John Domingue, Michael L. Brodie, and Frank Leymann. Semantic Web Services, Part 1. *IEEE Intelligent Systems*, 22(5):12–17, 2007.

[180] David Martin, John Domingue, Amit P. Sheth, Steve Battle, Katia P. Sycara, and Dieter Fensel. Semantic Web Services, Part 2. *IEEE Intelligent Systems*, 22(6):8–15, 2007.

[181] Davin Martin, Mark Burstein, Jerry Hobbs, Ora Lassila, Drew McDermott, Sheila McIlraith, Srini Narayanan, Massimo Paolucci, Bijan Parsia, Terry Payne, Evren Sirin, Naveen Srinivasan, and Katia Sycara. *OWL-S: Semantic Markup for Web Services*. W3C Member Submission, November 2004. http://www.w3.org/Submission/OWL-S/, access at 2009-06-12.

[182] Deborah L. McGuinness and Frank van Harmelen, editors. *OWL Web Ontology Language Overview*. W3C Recommendation, February 2004. http://www.w3.org/TR/owl-features/, access at 2010-02-19.

[183] Sheila A. McIlraith, Tran Cao Son, and Honglei Zeng. Semantic Web Services. *IEEE Intelligent Systems*, 16(2):46–53, 2001.

[184] Brahim Medjahed, Athman Bouguettaya, and Ahmed K. Elmagarmid. Composing Web services on the Semantic Web. *The VLDB Journal*, 12(4):333–351, 2003.

[185] André Miede, Michael Niemann, Stefan Schulte, Julian Eckert, Aneta Kabzeva, Nicolas Repp, and Ralf Steinmetz. Selected Topics in Service Engineering and Management for Enterprise Systems. In *Inaugural Workshop on Enterprise Systems Research in MIS (Pre-ICIS 2008)*, 2008.

[186] George A. Miller. WordNet: a lexical database for English. *Communications of the ACM*, 38(11): 39–41, 1995.

[187] Tom M. Mitchell. *Machine Learning*. McGraw-Hill, New York, NY, USA, 1997.

[188] Nilo Mitra and Yves Lafon, editors. *SOAP Version 1.2 Part 0: Primer (Second Edition)*. W3C Recommendation, 2007. `http://www.w3.org/TR/soap12-part0/`, access at 2010-02-16.

[189] James Munkres. Algorithms for the Assignment and Transportation Problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38, 1957.

[190] Anthony Nadalin, Chris Kaler, Ronald Monzillo, and Phillip Hallam-Baker, editors. *Web Services Security: SOAP Message Security 1.1 (WS-Security 2004)*. OASIS Standard incorporating Approved Errata, November 2006. `http://docs.oasis-open.org/wss/v1.1/wss-v1.1-spec-errata-os-SOAPMessageSecurity.pdf`, access at 2010-02-16.

[191] Farrukh Najmi and Joseph Chiusano, editors. *ebXML Registry profile for Web Services. Version 1.0 Draft 3*. Draft OASIS Profile, September 2005. `http://www.oasis-open.org/committees/download.php/14756/regrep-ws-profile-1.0-draft3.pdf`, access at 2010-03-05.

[192] Daniele Nardi and Ronald J. Brachman. An Introduction to Description Logics. In Franz Baader, Diego Calvanese, Deborah L. McGuiness, Daniele Nardi, and Peter F. Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation and Applications*, chapter 1, pages 5–44. Cambridge University Press, New York, NY, USA, 1. edition, 2003.

[193] Richi Nayak and Bryan Lee. Web Service Discovery with additional Semantics and Clustering. In *2007 IEEE / WIC / ACM International Conference on Web Intelligence (WI 2007)*, pages 555–558. IEEE Computer Society, Washington, DC, USA, 2007.

[194] Konstantinos A. Nedas. *Munkres' (Hungarian) Algorithm*. 2005. `http://konstantinosnedas.com/dev/soft/munkres.htm`, access at 2010-01-10.

[195] Jörg Nitzsche, Tammo van Lessen, Dimka Karastoyanova, and Frank Leymann. BPEL for Semantic Web Services (BPEL4SWS). In *3rd International Workshop on Agents and Web Services in Distributed Environments (AWeSome'07) at On the Move to Meaningful Internet Systems (OTM 2007)*, volume 4805 of *Lecture Notes in Computer Science*, pages 179–188. Springer, Berlin Heidelberg, 2007.

[196] OASIS ebXML Joint Committee. *The Framework for eBusiness*. An OASIS White Paper, April 2006. `http://www.oasis-open.org/committees/download.php/17817/ebxmljc-WhitePaper-wd-r02-en.pdf`, last access at 2010-03-05.

[197] Daniel Oberle, Nadeem Bhatti, Saartje Brockmans, Michael Niemann, and Christian Janiesch. Countering Service Information Challenges in the Internet of Services. *Business & Information Systems Engineering*, 1(5):370–390, 2009.

[198] Swapna A. Oundhakar, Kunal Verma, Kaarthik Sivashanmugam, Amit P. Sheth, and John A. Miller. Discovery of Web Services in a Multi-Ontology and Federated Registry Environment. *International Journal of Web Service Research*, 2(3):1–32, 2005.

[199] Michael Pantazoglou and Aphrodite Tsalgatidou. The Unified Service Query Language. Technical report, National and Kapodistrian University of Athens, Greece, 2009.

[200] Massimo Paolucci, Takahiro Kawamura, Terry R. Payne, and Katia P. Sycara. Importing the Semantic Web in UDDI. In *International Workshop on Web Services, E-Business, and the Semantic Web (WES 2002) in connection with The 14th Conference on Advanced Information Systems Engineering (CAiSE 2002)*, volume 2512 of *Lecture Notes in Computer Science*, pages 225–236. Springer, Berlin Heidelberg, 2002.

[201] Massimo Paolucci, Takahiro Kawamura, Terry R. Payne, and Katia P. Sycara. Semantic Matching of Web Services Capabilities. In *The Semantic Web: First International Semantic Web Conference (ISWC 2002)*, volume 2342 of *Lecture Notes in Computer Science*, pages 333–347. Springer, Berlin Heidelberg, 2002.

[202] Apostolos Papageorgiou, Bastian Leferink, Julian Eckert, Nicolas Repp, and Ralf Steinmetz. Bridging the Gaps towards Structured Mobile SOA. In *7th International Conference on Advances in Mobile Computing & Multimedia (MoMM2009)*, pages 288–294. ACM, New York, NY, USA, 2009.

[203] Michael P. Papazoglou. Service-Oriented Computing: Concepts, Characteristics and Directions. In *Fourth International Conference on Web Information Systems Engineering (WISE 2003)*, pages 3–12. IEEE Computer Society, Washington, DC, USA, 2003.

[204] Michael P. Papazoglou. *Web Services: Principles and Technology*. Pearsor Education Limited, Harlow, England, 2008.

[205] Michael P. Papazoglou and Dimitrios Georgakopoulos. Service-Oriented Computing: Introduction. *Communications of the ACM*, 46(10):24–28, 2003.

[206] Michael P. Papazoglou, Paolo Traverso, Schahram Dustdar, Frank Leymann, and Bernd J. Krämer. Service-Oriented Computing Research Roadmap. In *Service Oriented Computing (SOC)*, number 05462 in Dagstuhl Seminar Proceedings, pages 38–45. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany, 2006.

[207] Michael P. Papazoglou, Paolo Traverso, Schahram Dustdar, and Frank Leymann. Service-Oriented Computing: State of the Art and Research Challenges. *IEEE Computer Magazine*, 40(11):38–45, 2007.

[208] Mike P. Papazoglou and Willem-Jan van den Heuvel. Service oriented architectures: approaches, technologies and research issues. *The VLDB Journal*, 16(3):389–415, 2007.

[209] Abhijit Patil, Swapna Oundhakar, Amit P. Sheth, and Kunal Verma. METEOR-S Web Service Annotation Framework. In *13th International World Wide Web Conference (WWW 2004)*, pages 553–562. ACM, New York, NY, USA, 2004.

[210] Cary Pennington, Jorge Cardoso, John A. Miller, Richard Scott Patterson, and Ivan Vasquez. Introduction to Web Services. In Jorge Cardoso, editor, *Semantic Web Services: Theory, Tools, and Applications*, Premier Reference Source, chapter VII, pages 134–154. IGI Global, Hershey, PA, USA, 2007.

[211] Thomi Pilioura and Aphrodite Tsalgatidou. Unified publication and discovery of semantic Web services. *ACM Transactions on The Web*, 3(3):1–44, 2009.

[212] Pierluigi Plebani and Barbara Pernici. URBE: Web Service Retrieval Based on Similarity Evaluation. *IEEE Transactions on Knowledge and Data Engineering*, 21(11):1629–1642, 2009.

[213] Martin F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[214] Chris Preist. A Conceptual Architecture for Semantic Web Services. In *The Semantic Web: Third International Semantic Web Conference (ISWC 2004)*, volume 3298 of *Lecture Notes in Computer Science*, pages 395–409. Springer, Berlin Heidelberg, 2004.

[215] Eric Prud'hommeaux and Andy Seaborne, editors. *SPARQL Query Language for RDF*. W3C Recommendation, January 2008. `http://www.w3.org/TR/rdf-sparql-query/`, access at 2010-03-10.

[216] Roy Rada, Hafedh Mili, Ellen Bicknell, and Maria Blettner. Development and Application of a Metric on Semantic Nets. *IEEE Transactions on Systems, Man and Cybernetics*, 19(1):17–30, 1989.

[217] Nicolas Repp, André Miede, Michael Niemann, and Ralf Steinmetz. WS-Re2Policy: A policy language for distributed SLA monitoring and enforcement. In *Third International Conference on Systems and Networks Communications (ICSNC 2008)*, pages 256–261. IEEE Computer Society, Washington, DC, USA, 2008.

[218] Nicolas Repp, Dieter Schuller, Melanie Siebenhaar, André Miede, Michael Niemann, and Ralf Steinmetz. On distributed SLA monitoring and enforcement in service-oriented systems. *International Journal On Advances in Systems and Measurements*, 2(1):33–43, 2009.

[219] Philip Resnik. Semantic Similarity in a Taxonomy: An Information-Based Measure and its Application to Problems of Ambiguity in Natural Language. *Journal of Artificial Intelligence Research*, 11: 95–130, 1999.

[220] Leonard Richardson and Sam Ruby. *Restful Web Services*. O'Reilly, Sebastopol, CA, USA, 2007. ISBN 9780596529260.

[221] Marco Luca Sbodio and Claude Moulin. SPARQL as an expression language for OWL-S. In *Workshop OWL-S Experiences and Future Developments at the 4th European Semantic Web Conference (ESWC 2007)*, 2007.

[222] Dieter Schuller, Apostolos Papageorgiou, Stefan Schulte, Julian Eckert, Nicolas Repp, and Ralf Steinmetz. Process Reliability in Service-Oriented Architectures. In *Third IEEE International Conference on Digital Ecosystems and Technologies (DEST 2009)*, pages 640–645. IEEE Computer Society, Washington, DC, USA, 2009.

[223] Stefan Schulte, Nicolas Repp, Dieter Schuller, and Ralf Steinmetz. From Web Service Policies to Automatic Deviation Handling: Supporting Semantic Description of Reactions to Policy Violations. In *Third IEEE International Conference on Semantic Computing (ICSC 2009)*, pages 312–317. IEEE Computer Society, Washington, DC, USA, 2009.

[224] W. Roy Schulte and Yefim F. Natis. "Service-Oriented" Architectures, Part 1. Technical Report SPA-401-068, Gartner Group, 1996.

[225] Fabrizio Sebastiani. Machine Learning in Automated Text Categorization. *ACM Computing Surveys*, 34(1):1–47, 2002.

[226] Nigel Shadbolt, Tim Berners-Lee, and Wendy Hall. The Semantic Web Revisited. *IEEE Intelligent Systems*, 21(3):96–101, 2006.

[227] Amit P. Sheth, Cartic Ramakrishnan, and Christopher Thomas. Semantics for the Semantic Web: The Implicit, the Formal and the Powerful. *International Journal on Semantic Web and Information Systems*, 1(1):1–18, 2005.

[228] Amit P. Sheth, Karthik Gomadam, and Jon Lathem. SA-REST: Semantically Interoperable and Easier-to-Use Services and Mashups. *IEEE Internet Computing*, 11(6):91–94, 2007.

[229] Dong Shou and Chi-Hung Chi. Effective Web Service Retrieval Based on Clustering. In *Fourth International Conference on Semantics, Knowledge and Grid (SKG2008)*, pages 469–472. IEEE Computer Society, Washington, DC, USA, 2008.

[230] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical OWL-DL reasoner. *Journal of Web Semantics*, 5(2):51–53, 2007.

[231] Kaarthik Sivashanmugam, Kunal Verma, Amit P. Sheth, and John A. Miller. Adding Semantics to Web Services Standards. In *International Conference on Web Services (ICWS 2003)*, pages 395–401. CSREA Press, 2003.

[232] Kaarthik Sivashanmugam, Kunal Verma, and Amit P. Sheth. Discovery of Web Services in a Federated Registry Environment. In *IEEE International Conference on Web Services (ICWS 2004)*, pages 270–278. IEEE Computer Society, Washington, DC, USA, 2004.

[233] Kaarthik Sivashanmugam, John A. Miller, Amit P. Sheth, and Kunal Verma. Framework for Semantic Web Process Composition. *International Journal of Electronic Commerce*, 9(2):71–106, 2005.

[234] Naveen Srinivasan, Massimo Paolucci, and Katia P. Sycara. An Efficient Algorithm for OWL-S Based Semantic Search in UDDI. In *First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004), Revised Selected Papers*, volume 3387 of *Lecture Notes in Computer Science*, pages 96–110. Springer, Berlin Heidelberg, 2004.

[235] Steffen Staab, Rudi Studer, Hans-Peter Schnurr, and York Sure. Knowledge Processes and Ontologies. *IEEE Intelligent Systems*, 16(1):26–34, 2001.

[236] Nathalie Steinmetz, Mick Kerrigan, Holger Lausen, Martin Tanler, and Adina Sirbu. Simplifying the Web Service Discovery Process. In *First International Workshop on Semantic Metadata Management and Applications (SeMMA 2008) at The 5th European Semantic Web Conference (ESWC 2008)*, volume 346 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

[237] Ralf Steinmetz and Klaus Wehrle, editors. *Peer-to-Peer Systems and Applications*, volume 3485 of *Lecture Notes in Computer Science*. Springer, Berlin Heidelberg, 2005.

[238] Michael Stollberg, Uwe Keller, Holger Lausen, and Stijn Heymans. Two-Phase Web Service Discovery Based on Rich Functional Descriptions. In *The Semantic Web: Research and Applications, 4th European Semantic Web Conference (ESWC 2007)*, volume 4519 of *Lecture Notes in Computer Science*, pages 99–113. Springer, Berlin Heidelberg, 2007.

[239] Sun Microsystems, Inc. *Effective SOA Deployment using an SOA Registry Repository. A Practical Guide*. September 2005. `http://www.sun.com/products/soa/registry/soa_registry_wp.pdf`, access at 2010-04-14.

[240] York Sure, Michael Erdmann, Jürgen Angele, Steffen Staab, Rudi Studer, and Dirk Wenke. OntoEdit: Collaborative Ontology Development for the Semantic Web. In *The Semantic Web: First International Semantic Web Conference (ISWC 2002)*, volume 2342 of *Lecture Notes in Computer Science*. Springer, Berlin Heidelberg, 2002.

[241] Katia P. Sycara, Matthias Klusch, Seth Widoff, and Jianguo Lu. Dynamic Service Matchmaking Among Agents in Open Information Environments. *SIGMOD Record*, 28(1):47–53, 1999.

[242] Katia P. Sycara, Seth Widoff, Matthias Klusch, and Jianguo Lu. Larks: Dynamic Matchmaking Among Heterogeneous Software Agents in Cyberspace. *Autonomous Agents and Multi-Agent Systems*, 5(2):173–203, 2002.

[243] Katia P. Sycara, Massimo Paolucci, Anupriya Ankolekar, and Naveen Srinivasan. Automated discovery, interaction and composition of Semantic Web services. *Journal of Web Semantics*, 1(1): 27–46, 2003.

[244] Tanveer Fathima Syeda-Mahmood, Gauri Shah, Rama Akkiraju, Anca-Andreea Ivan, and Richard Goodwin. Searching Service Repositories by Combining Semantic and Ontological Matching. In *2005 IEEE International Conference on Web Services (ICWS 2005)*, pages 13–20. IEEE Computer Society, Washington, DC, USA, 2005.

[245] Vuong Xuan Tran, Sutheera Puntheeranurak, and Hidekazu Tsuji. A New Service Matching Definition and Algorithm with SAWSDL. In *Third IEEE International Conference on Digital Ecosystems and Technologies (DEST 2009)*, pages 593–598. IEEE Computer Society, Washington, DC, USA, 2009.

[246] Vassileios Tsetsos, Christos Anagnostopoulos, and Stathes Hadjiefthymiades. On the Evaluation of Semantic Web Service Matchmaking Systems. In *Fourth IEEE European Conference on Web Services (ECOWS 2006), 4-6 December 2006, Zürich, Switzerland*, pages 255–264. IEEE Computer Society, 2006.

[247] Vassileios Tsetsos, Christos Anagnostopoulos, and Stathes Hadjiefthymiades. Semantic Web Service Discovery: Methods, algorithms, and tools. In Jorge Cardoso, editor, *Semantic Web Services: Theory, Tools, and Applications*, Premier Reference Source, chapter XI, pages 240–280. IGI Global, Hershey, PA, USA, 2007.

[248] Mathias Uslar, Fabian Grüning, and Sebastian Rohjans. A Use Case for Ontology Evolution and Interoperability: The IEC Utility Standards Reference Framework 62357. In Yannis Kalfoglou, editor, *Cases on Semantic Interoperability for Information Systems Integration: Practices and Applications*, chapter IX, pages 39–46. Information Science Reference (an imprint of IGI Global), Hershey, PA, USA, 2009.

[249] Asir S. Vedamuthu, David Orchard, Frederick Hirsch, Maryann Hondo, Prasad Yendluri, Toufic Boubez, and Ümit Yalçınalp. *Web Services Policy 1.5 – Framework*. W3C Recommendation, September 2007. `http://www.w3.org/TR/ws-policy/`, access at 2010-02-16.

[250] Kunal Verma, Kaarthik Sivashanmugam, Amit P. Sheth, Abhijit Patil, Swapna Oundhakar, and John Miller. METEOR-S WSDI: A Scalable P2P Infrastructure of Registries for Semantic Publication and Discovery of Web Services. *Journal of Information Technology and Management*, 6(1):17–39, 2005.

[251] Tomas Vitvar, Jacek Kopecký, Jana Viskova, and Dieter Fensel. WSMO-Lite Annotations for Web Services. In *The Semantic Web: Research and Applications, 5th European Semantic Web Conference (ESWC 2008)*, volume 5021 of *Lecture Notes in Computer Science*, pages 674–689. Springer, Berlin Heidelberg, 2008.

[252] Johanna Völker, Denny Vrandecic, York Sure, and Andreas Hotho. Learning Disjointness. In *The Semantic Web: Research and Applications, 4th European Semantic Web Conference (ESWC 2007)*, volume 4519 of *Lecture Notes in Computer Science*, pages 175–189. Springer, Berlin Heidelberg, 2007.

[253] Ellen M. Voorhees. The Philosophy of Information Retrieval Evaluation. In *Evaluation of Cross-Language Information Retrieval Systems: Second Workshop of the Cross-Language Evaluation Forum (CLEF 2001), Revised Papers*, volume 2406 of *Lecture Notes in Computer Science*, pages 355–370. Springer, Berlin Heidelberg, 2002.

[254] Ellen M. Voorhees and Donna Harman. Overview of the Seventh Text REtrieval Conference TREC-7. In *Seventh Text REtrieval Conference (TREC-7)*, pages 1–24. National Institute of Standards and Technology (NIST), 1998.

[255] Taowei David Wang. Gauging Ontologies and Schemas by Numbers. In *4th International Workshop on Evaluation of Ontologies for the Web (EON2006) at the 15th International World Wide Web Conference (WWW 2006)*, 2006.

[256] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Publishers, San Francisco, CA, USA, 2. edition, 2005.

[257] Jeffrey M. Wooldridge. *Introductory Econometrics: A Modern Approach*. Thomson South-Western, 2nd edition, 2003.

[258] Qi Yu, Xumin Liu, Athman Bouguettaya, and Brahim Medjahed. Deploying and managing Web services: issues, solutions, and directions. *The VLDB Journal*, 17(3):537–572, 2008.

[259] Liang-Jie Zhang. EIC Editorial: Introduction to the Knowledge Areas of Services Computing. *IEEE Transactions on Services Computing*, 1(2):62–74, 2008.

[260] Hai Zhuge and Jie Liu. Flexible retrieval of Web Services. *Journal of Systems and Software*, 70 (1–2):107–116, 2004.

[261] Justin Zobel. *Writing for Computer Science*. Springer, Berlin Heidelberg, 2nd edition, 2004.

List of Figures

List of Tables

List of Listings

List of Acronyms

AFP	Advertisement Functional Profile ... 101
AI	Artificial Intelligence ... 16
AP	Average Precision ... 54
API	Application Programming Interface ... 44
AQRT	Average Query Response Time ... 54
ATSM	Abstract Truncated Service Model ... 17
CIM	Common Information Model ... 65
COV4SWS.KOM	Coverage-based Matchmaking Approach for Semantic Web Services ... 27
DAML-S	DARPA Agent Markup Language for Services ... 17
DL	Description Logics ... 7
DoM	Degree of Match ... 4
ebXML	Electronic Business using Extensible Markup Language ... 4
GOL	Grid Operation Language ... 102
GUI	Graphical User Interface ... 100
IR	Information Retrieval ... 4
IRI	Internationalized Resource Identifier ... 17
ISO	International Organization for Standardization ... 154
iSPARQL	Imprecise SPARQL ... 71
JAXB	Java Architecture for XML Binding ... 96
JAXR	Java API for XML Registries ... 86
LARKS	Language for Advertisement and Request for Knowledge Sharing ... 65
LOG4SWS.KOM	Logic-based Matchmaking Approach for Semantic Web Services ... 27
MAD	Median Absolute Deviation ... 168
MC	Mutual Coverage ... 49
OASIS	Organization for the Advancement of Structured Information Standards ... 152
OLS	Ordinary Least Squares ... 35
OWL	Web Ontology Language ... 14
OWL 2	OWL 2 Web Ontology Language ... 155
OWL-S	Web Ontology Language for Web Services ... 3
P2P	Peer-to-Peer ... 10
PQL	Process Query Language ... 67
PS-WSDL	PYRAMID-S WSDL ... 101
QoS	Quality of Service ... 6
RDF	Resource Description Framework ... 14
RDFS	RDF Schema ... 17
REST	Representational State Transfer ... 11
RFP	Request Functional Profile ... 101
RIM	Registry Information Model ... 95

RP	R-Precision 54
RS	Registry Services and Protocols 154
S3 Contest	Annual International Contest S3 on Semantic Service Selection – Retrieval Performance Evaluation of Matchmakers for Semantic Web Services 28
SAWSDL	Semantic Annotations for WSDL and XML Schema 3
SME2	Semantic Web Service Matchmaker Evaluation Environment 115
SOA	Service-oriented Architecture 3
SOAP	originally defined as Simple Object Access Protocol 3
SOC	Service-oriented Computing 3
SPARQL	SPARQL Protocol and RDF Query Language 6
SQL	Structured Query Language 5
SVM	Support Vector Machine 69
SWRL	Semantic Web Rule Language 19
SWS	semantic Web service 3
SWS2QL	SWS Structured Query Language 7
TF-IDF	Term Frequency-Inverse Document Frequency 65
tModel	technical Model 79
UDDI	Universal Description, Discovery and Integration 6
URBE	Uddi Registry By Example 48
URI	Uniform Resource Identifier 17
URL	Uniform Resource Locator 80
USDL	Universal Service Description Language 115
USQL	Unified Service Query Language 101
UUID	Universally Unique IDentifier 80
WS-BPEL	Web Services Business Process Execution Language 11
W3C	World Wide Web Consortium 3
WADL	Web Application Description Language 115
WSDL	Web Service Description Language 3
WSDL-S	Web Service Semantics 17
WSML	Web Service Modeling Language 3
WSMO	Web Service Modeling Ontology 17
WSMX	Web Service Modeling eXecution Environment 102
XML	Extensible Markup Language 3
XSD	XML Schema Definition Language 11
XSLT	Extensible Stylesheet Language Transformation 81

Part IV
Appendix

A Standards

In Chapter 2, a number of standards and technologies have been introduced with respect to their intended usage and some conceptual and technical characteristics. In order to close the gaps between these introductions to Web service standards and the implementations from Chapters 3 and 4, the first appendix chapter presents a more detailed, technical introduction to these standards.

To start with, Section A.1 introduces technical aspects of the Web service formalisms applied in this thesis. Afterwards, the service registry standards UDDI and ebXML Registry are examined. In Section A.3, RDF and OWL, i.e., the metadata respectively ontology standards applied in this thesis, are presented. The section is completed by an introduction to Description Logics, as OWL DL is the applied OWL variant. Finally, SPARQL, which is the standard query language for the Semantic Web, is presented in Section A.4.

A.1 Web Service Formalisms

In the following, technical aspects of WSDL and SAWSDL will be presented (Section A.1.1). Afterwards, the mapping from WSDL 1.1 to 2.0 and WSDL to RDF is presented. This section ends with an introduction to the Abstract Truncated Service Model (ATSM), which is used as intermediary service formalism in this thesis.

A.1.1 Web Service Description Language and Semantic Annotations for WSDL and XML Schema

Listing A.1: XML Representation of WSDL 2.0 Description Component

```
<description                                                                    1
      targetNamespace=''xs:anyURI'' >                                          2
  <documentation />*                                                           3
  [ <import /> | <include /> ]*                                                4
  <types />?                                                                   5
  [ <interface /> | <binding /> | <service /> ]*                               6
</description>                                                                 7
```

Listing A.1 shows the XSD definition of WSDL 2.0 [57]; an example WSDL file from SAWSDL-TC can be found in Listing A.2. Apart from the XML declaration, the topmost element in any WSDL 2.0 declaration is the `description` element. `description` declares a container for WSDL 2.0 components and type definitions [57]. Like most of its subelements, `description` possesses an optional `documentation` element. The content of this documentation is not further defined, i.e., it could be machine-processable and/or human readable.

The `import` and `include` elements are also optional and allow to include element information from other service definitions. If these service definitions belong to the same namespace, the `include` element is used, otherwise `import`.

Figure 2.4 (Section 2.1.2) shows the main components of a WSDL document. As it can be seen, the service interface is defined using the `interface` element and made up from the interface itself and its optional `operation` subelements as well as `input` and `output` messages, which belong to a particular operation. Furthermore, an interface might contain an optional `fault` element, which describes the type of an event invoked if the normal execution of a message exchange is disrupted [57]. An operation might also contain `infault` and `outfault` elements referencing a type defined by the interface `fault` component. Each operation possesses a required `pattern` attribute which indicates a message exchange sequence. For example, if a service receives a message as *in*put and subsequently returns a response back to the client (*out*put), the *in-out*-pattern would be used [56].

Apart from the `interface` element, a WSDL 2.0 file might also contain `binding` and `service` elements. These elements define where a service is located and how it can be invoked using a particular message format. In the work at hand, bindings and endpoints are only of minor interest as the description, *what* a service does, is wrapped inside the interface. Hence, bindings and endpoints will not be investigated any further. For an overview of these elements, we refer to the according W3C recommendation [57].

In contrast, the *service signature* (cp. Section 2.1.2) is of very high importance, as the signature elements form the basis for most service matchmakers (cp. Section 3.5). The service signature is defined in the `types` section of a WSDL description and referenced by `input` and `output` elements. Message types can be inlined or imported into service descriptions. The actual chosen method is only of subordinated relevance for the work at hand. We assume that message types are directly inserted in a WSDL 2.0 document using the `types` element, which is a direct child of the WSDL `description` element. It is possible to define simple and complex message types. Top-level types need to be defined as single elements but might have a substructure.

Listing A.2 shows an example of a WSDL 2.0 specification where complex message types are defined using XSD (e.g., Lines 5–13). Although it is possible to define message types using different schema languages, XSD is the de facto standard language for message types in WSDL 2.0 as it is natively supported [32]. As it can be seen from the example, the `input` and `output` elements refer to single XSD elements, which in turn can be made of sequences, restrictions, and further elements. For an overview of XSD, we refer to the according W3C recommendation [78].

Furthermore, the example from Listing A.2 shows how interfaces, operations, inputs, and outputs are interrelated to each other and how XSD elements are semantically annotated using SAWSDL's `modelReference` attribute. Model references might be used to annotate XSD type definitions, element declarations, and attribute declarations, and WSDL interfaces, operations, and faults. Most notably, inputs and outputs are not directly annotated but indirectly semantically enriched using the type definitions [32]. Furthermore, attributes `liftingSchemaMapping` and `loweringSchemaMapping` are also defined in SAWSDL (cp. Section 2.2.3) but not applied in this thesis.

A.1.2 Mapping WSDL to RDF with WS2RDF.KOM

For the application of SPARQL queries in the UDDI enhancement presented in Section 4.3, it is necessary to map WSDL-based service descriptions to RDF. Basically, this can be done following the official SAWSDL and WSDL 2.0 to RDF mappings specified in [79, 143]. In the software component *WS2RDF.KOM*, this mapping is applied and enhanced in order to facilitate not only a mapping from WSDL 2.0 to RDF, but also from WSDL 1.1, and, most notably, of embedded XSD type definitions.

The conversion is based on an extension of the corresponding XSLT stylesheet provided by the W3C[1], where *Saxon*[2] – an open source XSLT processor – is utilized. Unfortunately, the prototype converter by W3C was error-prone and had to be adapted as described in the following. The new XSLT can be found at `http://www.kom.tu-darmstadt.de/~schulte/wsdl11to20.xsl`.

Once a WSDL 2.0 source is available, it is fed into an implementation of the W3C mappings by Kopecký[3], based on *Woden4SAWSDL*, a framework for processing SAWSDL files, which is also applied in the matchmaking respectively registry implementations presented in Chapters 3 and 4. Woden4SAWSDL provides a complete object model for SAWSDL documents and allows to ensure compliance with the WSDL 2.0 specification, process the document structure with its individual elements, and access attributes through explicit methods.

In WSDL 2.0, data structures are usually represented using XSD (cp. Section 2.1.2). However, there is no official mapping from XSD to RDF. Hence, it was necessary to devise and implement such a mapping. While our solution is proprietary in nature, it adheres to the principal conventions described in the W3C's WSDL 2.0 to RDF mapping [143].

While Kopecký's implementation initially provided a simple text output in N3 serialization, we have adapted it in order to utilize the *Jena Semantic Web Framework* [51] for holding the output stream. This way, various serializations of the resulting RDF output may be generated. Furthermore, this process ensures implicit verification of the RDF stream.

[1] `http://www.w3.org/2006/02/wsdl11to20.xsl`
[2] `http://saxon.sourceforge.net/`
[3] `http://jacek.cz/tmp/wsdl2rdf-src.jar`

Listing A.2: WSDL Service (Example)

```
<?xml version=''1.0'' encoding=''utf-8''?>
<wsdl20:description ...>
   <wsdl20:types>
      <xsd:schema ...>
         <xsd:complexType name=''OrganizationType'' sawsdl:modelReference=''ont/portal.owl#Organization''>
            <xsd:sequence>
               <xsd:element name=''has-sub-unit'' type=''Organization-Unit''/>
               <xsd:element name=''organization-part-of'' type=''OrganizationType''/>
               <xsd:element name=''affiliated-person'' type=''Affiliated-Person''/>
               <xsd:element name=''headed-by'' type=''Affiliated-Person''/>
               <xsd:element name=''has-size'' type=''OrganizationOrganization-Size''/>
            </xsd:sequence>
         </xsd:complexType>
         <xsd:complexType name=''Affiliated-Person''
                          sawsdl:modelReference=''ont/portal.owl#Affiliated-Person''>
            <xsd:sequence>
               <xsd:element name=''has-affiliation'' type=''OrganizationType''/>
            </xsd:sequence>
         </xsd:complexType>
         <xsd:simpleType name=''DestinationType'' sawsdl:modelReference=''ont/travel.owl#Destination''>
            <xsd:restriction base=''xsd:string''/>
         </xsd:simpleType>
         <xsd:simpleType name=''OrganizationOrganization-Size'' sawsdl:modelReference=''''>
            <xsd:restriction base=''Organization-Size''>
            </xsd:restriction>
         </xsd:simpleType>
         <xsd:simpleType name=''Organization-Size''
                         sawsdl:modelReference=''ont/portal.owl#Organization-Size''>
            <xsd:restriction base=''xsd:string''>
            </xsd:restriction>
         </xsd:simpleType>
         <xsd:simpleType name=''SurfingType'' sawsdl:modelReference=''ont/travel.owl#Surfing''>
            <xsd:restriction base=''xsd:string''/>
         </xsd:simpleType>
         <xsd:simpleType name=''Organization-Unit''
                         sawsdl:modelReference=''ont/portal.owl#Organization-Unit''>
            <xsd:restriction base=''xsd:string''/>
         </xsd:simpleType>
         <xsd:element name=''get_DESTINATIONRequest''>
            <xsd:complexType>
               <xsd:sequence>
                  <xsd:element name=''_ORGANIZATION'' type=''tns:OrganizationType''/>
                  <xsd:element name=''_SURFING'' type=''tns:SurfingType''/>
               </xsd:sequence>
            </xsd:complexType>
         </xsd:element>
         <xsd:element name=''get_DESTINATIONResponse''>
            <xsd:complexType>
               <xsd:sequence>
                  <xsd:element name=''_DESTINATION'' type=''tns:DestinationType''/>
               </xsd:sequence>
            </xsd:complexType>
         </xsd:element>
      </xsd:schema>
   </wsdl20:types>
   <wsdl20:interface name=''SurfingorganizationDestinationSoap''>
      <wsdl20:operation name=''get_DESTINATION'' pattern=''http://www.w3.org/ns/wsdl/in-out''>
         <wsdl20:input element=''tns:get_DESTINATIONRequest''/>
         <wsdl20:output element=''tns:get_DESTINATIONResponse''/>
      </wsdl20:operation>
   </wsdl20:interface>
</wsdl20:description>
```

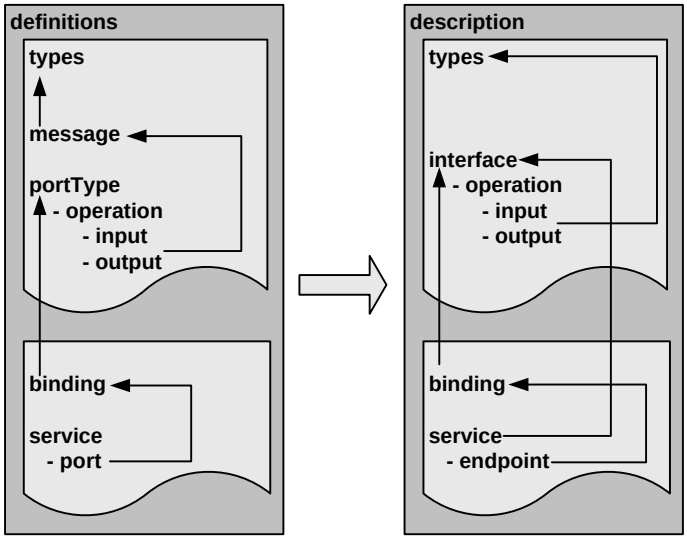


Figure A.1: WSDL 1.1 in Comparison with WSDL 2.0 (taken from [116])

Figure A.1 shows the different components of WSDL 1.1 respectively WSDL 2.0. Although there are some differences between the WSDL versions, both provide an abstract and a concrete description of a Web service. A major difference between the WSDL versions are the missing `message` elements in WSDL 2.0. Instead, `input` and `output` messages are directly defined using XSD. Here, the prototype converter of W3C featured an error, which prevented the correct transformation of `message` elements into XSD elements. To resolve this issue, the conversion workflow was changed as described in the following. In general, the conversion of a WSDL 1.1 to a WSDL 2.0 file follows the XML tree structure used in WSDL 1.1, i.e., the WSDL 1.1 file is processed from top to bottom and each WSDL 1.1 component is mapped to its corresponding WSDL 2.0 component. More precisely:

1. The WSDL 1.1 `definitions` element is mapped to a WSDL 2.0 `description` element. All namespaces needed in the WSDL 2.0 file are copied.

2. WSDL 1.1 `types` elements are copied to the WSDL 2.0 file. Normally, a `types` element is made up from an XSD `schema` element. In the prototype converter, the `schema` element is simply copied; in the solution at hand, a new `schema` element is created possessing a `targetNamespace` attribute. The value of this attribute is set to be the same as defined in the top WSDL 2.0 `description` element in order to avoid inconsistent declarations and usage of XSD elements. Afterwards, the declared elements in the WSDL 1.1 file are mapped from `message` (WSDL 1.1) to `schema` (WSDL 2.0) elements. Therefore, an XSD `complexType` is created for each `message` element. Every `complexType` element features a `sequence` child element. For every `messagePart` element of a `message` element, a new XSD `element` with the according `messagePart`'s name and type is created.

3. In the next step, `portType` elements are mapped to WSDL 2.0 `interface` elements. This step is quite straightforward, as the only elements to be adapted are `operations`. In WSDL 2.0, the `pattern` attribute is mandatory. Here, for every Web service a common pattern is applied, namely the "in-out" pattern [32]. The prototype converter has a mechanism to extract the pattern from the WSDL 1.1 source; however, tests have shown that it is not necessarily leading to correct results. In WSDL 1.1, an `input` or `output` has got the attribute `message`, pointing to the `message` element declaration. As WSDL 2.0 abstains from `message` elements, the WSDL 1.1 messages attributes have to be replaced by `element` attributes.

4. The remaining conversion is analog to the W3C converter – this concerns the mapping of the `service`, `binding`, and `port` elements.

Mapping from WSDL 2.0 / SAWSDL and XSD to RDF

The mapping from WSDL 2.0 to RDF respectively SAWSDL to RDF is part of the official specifications by W3C [79, 143]. Its main contribution is a standard RDF and OWL vocabulary we will recapitulate in short: An OWL ontology for WSDL defines the core WSDL components, i.e., the `description` element, `interface`, `operation`, `binding`, and `service` element as well as their subelements. An example WSDL file and its RDF counterpart are depicted in Listing A.2 and Listing A.3 respectively. As can be seen, each major component in the WSDL document is represented as an individual RDF subject and points to its child components by means of a specific RDF predicate.

As previously mentioned, the W3C's mapping from WSDL 2.0 to RDF does not specify a transformation for XSD element declarations and type definitions. Accordingly, we have implemented a proprietary mapping, which follows the principles outlined by the W3C's WSDL to RDF mapping with respect to naming and representation of relations between components. The procedure in the mapping is explained based on Listing A.2, which depicts the `surfingorganization_destination_service.wsdl` file from SAWSDL-TC, and the resulting output, depicted in Listing A.3. The service's single operation, `get_DESTINATION`, specifies the `get_DESTINATIONResponse` XSD element as output. In the corresponding RDF representation, this is reflected in Lines 48 and 49, where the predicate `elementDeclaration` points to an object `urn:uuid:9a49f440-da24-4a9a-9862-b17d247be8c3`. That object, which is further specified in Lines 76–83, establishes a link towards an `elementDeclaration` component specified in Lines 85–88, namely `get_DESTINATIONResponse`. In Lines 90–115, the underlying structure of the `get_DESTINATIONResponse` XSD element is represented. As can be seen from the WSDL source file, the element is defined as `complexType`, with a `sequence` (Lines 95–98) embedded in it. That sequence contains one element by the name of `_DESTINATION` (Lines 100–103), which points to a complex type with the name `DestinationType` (Lines 105–110), which in turn is addressed by a restriction (Lines 112–115).

Listing A.3: RDF Representation of SAWSDL (Example)

```
@prefix tns: <http://dmas.dfki.de/axis/services/SurfingorganizationDestination#>                                  1
                                                                                                                  2
<tns:wsdl.description()>                                                                                          3
      a       <http://www.w3.org/ns/wsdl-rdf#Description> ;                                                       4
      <http://www.w3.org/ns/wsdl-rdf#binding>                                                                     5
              <tns:wsdl.binding(SurfingorganizationDestinationSoapBinding)> ;                                     6
      <http://www.w3.org/ns/wsdl-rdf#interface>                                                                   7
              <tns:wsdl.interface(SurfingorganizationDestinationSoap)> ;                                          8
      <http://www.w3.org/ns/wsdl-rdf#service>                                                                     9
              <tns:wsdl.service(SurfingorganizationDestinationService)> .                                         10
                                                                                                                  11
<tns:wsdl.service(SurfingorganizationDestinationService)>                                                         12
      a       <http://www.w3.org/ns/wsdl-rdf#Service> ;                                                           13
      <http://www.w3.org/2000/01/rdf-schema#label>                                                                14
              ''SurfingorganizationDestinationService'' ;                                                         15
      <http://www.w3.org/ns/wsdl-rdf#endpoint>                                                                    16
              <tns:wsdl.endpoint(SurfingorganizationDestinationService/SurfingorganizationDestinationSoap)> ;     17
      <http://www.w3.org/ns/wsdl-rdf#implements>                                                                  18
              <tns:wsdl.interface(SurfingorganizationDestinationSoap)> .                                          19
                                                                                                                  20
<tns:wsdl.interface(SurfingorganizationDestinationSoap)>                                                          21
      a       <http://www.w3.org/ns/wsdl-rdf#Interface> ;                                                         22
      <http://www.w3.org/2000/01/rdf-schema#label>                                                                23
              ''SurfingorganizationDestinationSoap'' ;                                                            24
      <http://www.w3.org/ns/wsdl-rdf#interfaceOperation>                                                          25
              <tns:wsdl.interfaceOperation(SurfingorganizationDestinationSoap/get_DESTINATION)> .                 26
                                                                                                                  27
<tns:wsdl.interfaceOperation(SurfingorganizationDestinationSoap/get_DESTINATION)>                                 28
      a       <http://www.w3.org/ns/wsdl-rdf#InterfaceOperation> ;                                                29
      <http://www.w3.org/2000/01/rdf-schema#label>                                                                30
              ''get_DESTINATION'' ;                                                                               31
      <http://www.w3.org/ns/wsdl-rdf#interfaceMessageReference>                                                   32
              <tns:wsdl.interfaceMessageReference(SurfingorganizationDestinationSoap/get_DESTINATION/Out)> , <    33
                   tns:wsdl.interfaceMessageReference(SurfingorganizationDestinationSoap/get_DESTINATION/In)>
                   ;
      <http://www.w3.org/ns/wsdl-rdf#messageExchangePattern>                                                      34
              <http://www.w3.org/ns/wsdl/in-out> .                                                                35
                                                                                                                  36
<tns:wsdl.interfaceMessageReference(SurfingorganizationDestinationSoap/get_DESTINATION/In)>                       37
      a       <http://www.w3.org/ns/wsdl-rdf#InterfaceMessageReference> , <http://www.w3.org/ns/wsdl-rdf#        38
           InputMessage> ;
      <http://www.w3.org/ns/wsdl-rdf#elementDeclaration>                                                          39
              <urn:uuid:1105a2eb-2531-4da7-9586-2dee4a1f006b> ;                                                   40
```

```
      <http://www.w3.org/ns/wsdl-rdf#messageContentModel>                                                          41
              <http://www.w3.org/ns/wsdl-rdf#ElementContent> ;                                                     42
      <http://www.w3.org/ns/wsdl-rdf#messageLabel>                                                                 43
              <http://www.w3.org/ns/wsdl/in-out#In> .                                                              44
                                                                                                                   45
<tns:wsdl.interfaceMessageReference(SurfingorganizationDestinationSoap/get_DESTINATION/Out)>                       46
      a       <http://www.w3.org/ns/wsdl-rdf#InterfaceMessageReference> , <http://www.w3.org/ns/wsdl-rdf#          47
          OutputMessage> ;
      <http://www.w3.org/ns/wsdl-rdf#elementDeclaration>                                                           48
              <urn:uuid:9a49f440-da24-4a9a-9862-b17d247be8c3> ;                                                    49
      <http://www.w3.org/ns/wsdl-rdf#messageContentModel>                                                          50
              <http://www.w3.org/ns/wsdl-rdf#ElementContent> ;                                                     51
      <http://www.w3.org/ns/wsdl-rdf#messageLabel>                                                                 52
              <http://www.w3.org/ns/wsdl/in-out#Out> .                                                             53
                                                                                                                   54
<tns:wsdl.binding(SurfingorganizationDestinationSoapBinding)>                                                      55
      a       <http://www.w3.org/ns/wsdl/soap> , <http://www.w3.org/ns/wsdl-rdf#Binding> ;                         56
      <http://www.w3.org/2000/01/rdf-schema#label>                                                                 57
              ''SurfingorganizationDestinationSoapBinding'' ;                                                      58
      <http://www.w3.org/ns/wsdl-rdf#bindingOperation>                                                             59
              <tns:wsdl.bindingOperation(SurfingorganizationDestinationSoapBinding/get_DESTINATION)> ;             60
      <http://www.w3.org/ns/wsdl-rdf#binds>                                                                        61
              <tns:wsdl.interface(SurfingorganizationDestinationSoap)> ;                                           62
      <http://www.w3.org/ns/wsdl/http#defaultQueryParameterSeparator>                                              63
                                                                                                                   64
              ''&'' ;                                                                                              65
      <http://www.w3.org/ns/wsdl/soap#protocol>                                                                    66
              <http://www.w3.org/2006/01/soap11/bindings/HTTP/> ;                                                  67
      <http://www.w3.org/ns/wsdl/soap#version>                                                                     68
              ''1.1'' .                                                                                            69
                                                                                                                   70
<tns:wsdl.bindingOperation(SurfingorganizationDestinationSoapBinding/get_DESTINATION)>                             71
      a       <http://www.w3.org/ns/wsdl-rdf#BindingOperation> ;                                                   72
      <http://www.w3.org/ns/wsdl-rdf#binds>                                                                        73
              <tns:wsdl.interfaceOperation(SurfingorganizationDestinationSoap/get_DESTINATION)> .                  74
                                                                                                                   75
<urn:uuid:9a49f440-da24-4a9a-9862-b17d247be8c3>                                                                    76
      a       <http://www.w3.org/ns/wsdl-rdf#QName> ;                                                              77
      <http://www.kom.tu-darmstadt.de/ns/xsd-rdf#describedBy>                                                      78
              <tns:xsd.elementDeclaration(get_DESTINATIONResponse)> ;                                              79
      <http://www.w3.org/ns/wsdl-rdf#localName>                                                                    80
              ''get_DESTINATIONResponse'' ;                                                                        81
      <http://www.w3.org/ns/wsdl-rdf#namespace>                                                                    82
              <http://dmas.dfki.de/axis/services/SurfingorganizationDestination> .                                 83
                                                                                                                   84
<tns:xsd.elementDeclaration(get_DESTINATIONResponse)>                                                              85
      a       <http://www.kom.tu-darmstadt.de/ns/xsd-rdf#element> ;                                                86
      <http://www.kom.tu-darmstadt.de/ns/xsd-rdf#complexType>                                                      87
              <tns:xsd.elementDeclaration(a50497ee-d662-4142-b20d-0f9bbb4e1e48)> .                                  88
                                                                                                                   89
<tns:xsd.elementDeclaration(a50497ee-d662-4142-b20d-0f9bbb4e1e48)>                                                 90
      a       <http://www.kom.tu-darmstadt.de/ns/xsd-rdf#complexType> ;                                            91
      <http://www.kom.tu-darmstadt.de/ns/xsd-rdf#sequence>                                                         92
              <tns:xsd.sequence(00e5f7e4-aa4d-4645-b300-32bd610206a7)> .                                           93
                                                                                                                   94
<tns:xsd.sequence(00e5f7e4-aa4d-4645-b300-32bd610206a7)>                                                           95
      a       <http://www.kom.tu-darmstadt.de/ns/xsd-rdf#sequence> ;                                               96
      <http://www.kom.tu-darmstadt.de/ns/xsd-rdf#element>                                                          97
              <tns:xsd.elementDeclaration(_DESTINATION)> .                                                         98
                                                                                                                   99
<tns:xsd.elementDeclaration(_DESTINATION)>                                                                         100
      a       <http://www.kom.tu-darmstadt.de/ns/xsd-rdf#element> ;                                                101
      <http://www.kom.tu-darmstadt.de/ns/xsd-rdf#type>                                                             102
              <tns:xsd.elementDeclaration(DestinationType)> .                                                      103
                                                                                                                   104
<tns:xsd.elementDeclaration(DestinationType)>                                                                      105
      a       <http://www.kom.tu-darmstadt.de/ns/xsd-rdf#simpleType> ;                                             106
      <http://www.kom.tu-darmstadt.de/ns/xsd-rdf#restriction>                                                      107
              <tns:xsd.restriction(b5f51868-da85-4b39-860a-b3c693bc9b11)> ;                                        108
      <http://www.w3.org/ns/sawsdl#modelReference>                                                                 109
              <http://127.0.0.1/ontology/travel.owl#Destination> .                                                 110
                                                                                                                   111
<tns:xsd.restriction(b5f51868-da85-4b39-860a-b3c693bc9b11)>                                                        112
      a       <http://www.kom.tu-darmstadt.de/ns/xsd-rdf#restriction> ;                                            113
      <http://www.kom.tu-darmstadt.de/ns/xsd-rdf#base>                                                             114
               <http://www.w3.org/2001/XMLSchema#string> .                                                         115
```

A.1.3 Abstract Truncated Service Model

Instead of applying the matchmakers from Chapter 3 or the discovery framework from Section 4.4 to SAWSDL-based service descriptions, we make use of ATSM. ATSM constitutes a lightweight model that solely embodies the abstract parts of service descriptions that are essential in the discovery process. It also allows to represent existing SWS description standards, namely SAWSDL and OWL-S, in one common model. For that purpose, common components and attributes in Web service descriptions have been identified.

The need for such a model is primarily triggered by practical reasons. The effort and overhead that is contained in the deserialization and general processing of full-featured service descriptions as given by the previously mentioned standards is fairly high. In contrast, a lightweight service model allows faster and less memory-intensive processing in the course of matchmaking. Another reason is the existence of different standards, which usually requires the implementation of different variants of a matchmaker, each accounting for the specific characteristics of the underlying description language. For instance, Klusch et al. have published different versions of the "MX"-family of matchmakers for WSMO/WSML, OWL-S, and SAWSDL (cp. Section 3.5). In contrast, our approach allows to wrap description languages, facilitating a matchmaker implementation to operate on a common metamodel.

In the following, ATSM will be introduced regarding its components, the mapping from SAWSDL and OWL-S to ATSM, and its implementation and serialization.

Components in ATSM and Mapping from OWL-S and (SA)WSDL to ATSM

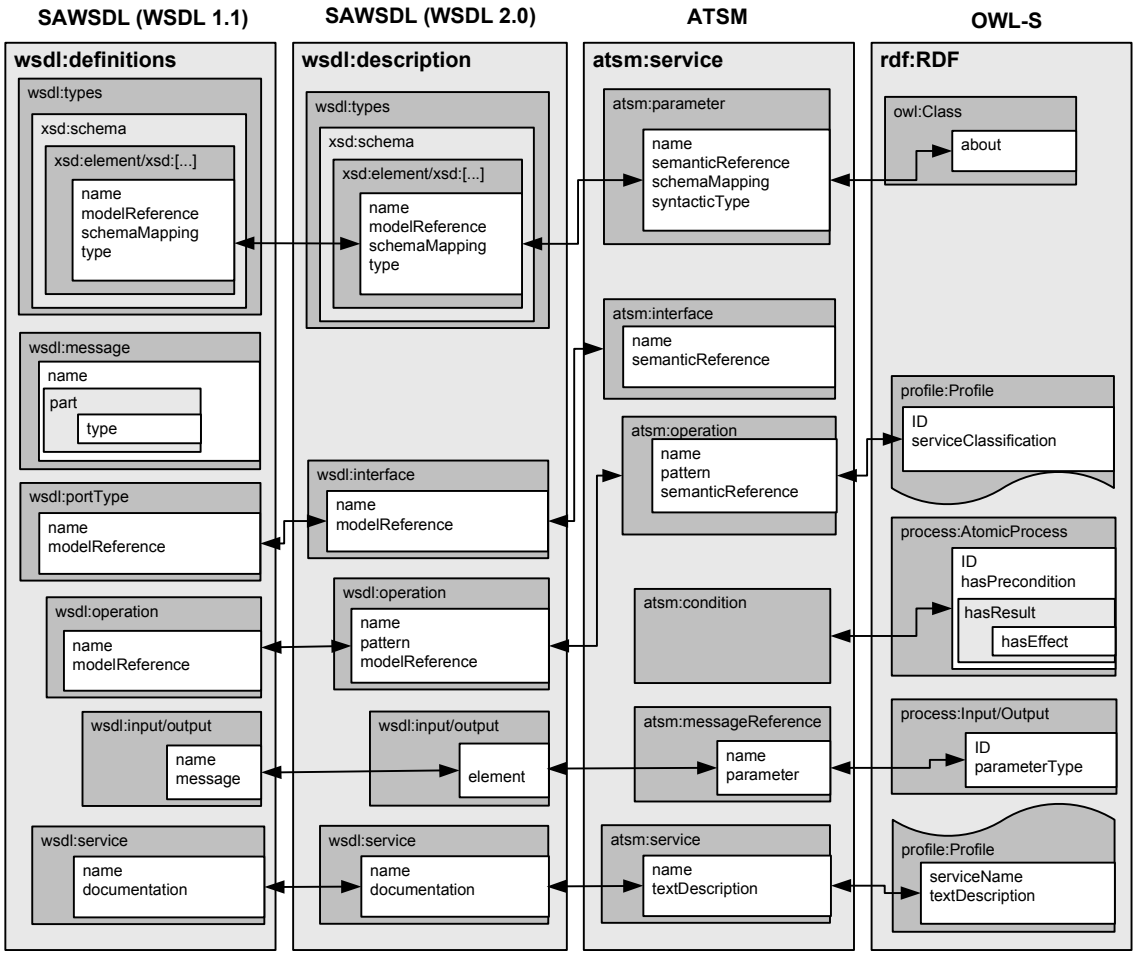


Figure A.2: Mapping from OWL-S and SAWSDL to ATSM

ATSM shares a high similarity with the WSDL component model (cp. Section 2.1). An overview of the ATSM component model is provided in Figure A.2. A service description in ATSM can essentially be interpreted as a tree: Each `service` component may contain 0 or more `interfaces`, which each subsume 0 or more `operations`. Each operation links to 0 or more `messageReferences`, which in turn are composed by 0 or more `parameters`. Message reference, in this context, is a generic term for `inputs` and `outputs`. Furthermore, we consider `conditions` (i.e., preconditions and effects) for each operation. However, conditions should be primarily recognized as a placeholder, as the format of preconditions and effects has not been definitely defined regarding SAWSDL and OWL-S.

Apart from conditions, each component has a set of attributes. Regarding parameters, this includes the component's name (generally encoded as qualified name), a set of URIs that point to associated semantic concepts, a URI describing its syntactic type, and a set of URIs pointing to lifting or lowering schema mappings for automatic mediation between concrete data and ontological concepts.

The choice of attributes is based on the following assumptions: Semantic annotations (or references) provide the most precise source of information in matchmaking, e.g., in traditional, logic-based subsumption reasoning. If they are not available, component names allow to derive textual similarity using standard methods from IR. Syntactic information (for instance, the data type) is of importance on the level of inputs and outputs and can be used in automatic mediation between services.

To allow for the representation of existing services in ATSM, we have devised a mapping that defines correspondencies between components and their attributes in SAWSDL and OWL-S service descriptions and an ATSM document (cp. Figure A.2). Because of the high similarity between the WSDL and ATSM component models, the mapping from WSDL interface, operation, input, and output components can be conducted to the respective equivalents in ATSM in a straightforward manner. The `modelReference` attribute has been renamed to `semanticReference`, assuming that annotations correspond to some semantic concept in an ontology.

XSD elements are mapped onto ATSM `parameter` components. An ATSM parameter holds both semantic and syntactic information, as well as the schema mappings that may be defined using SAWSDL annotations. The structure of the XSD definition is eliminated in the mapping by generating a "flat" parameter structure: `sequence` and `complexType` elements are recursively resolved in a top-down manner, resulting in a set of ATSM `parameters`. In detail, the mapping process works as follows: If an XSD element does not specify a type, the underlying structure – commonly a sequence – is further evaluated. For each contained XSD element, the process is recursively repeated. If an element refers to a `simpleType` or `complexType`, the type is transformed into an ATSM `parameter` component. Additionally, for a `complexType`, the structure is broken down and recursively processed as previously outlined. The process continues up to a specified recursive depth or until a loop is detected.

The process is illustrated along the lines of the schema definition contained in the `surfing-organization_destination_service.wsdl` file from SAWSDL-TC (cp. Listing A.2); an excerpt of the resulting ATSM document is depicted in Listing A.4. The only operation in the service, `get_DESTINATION`, specifies the element `get_DESTINATIONRequest` as input. This element consists of a sequence, containing the element `_ORGANIZATION` (the second element, `_SURFING`, is omitted for the purpose of simplicity), which points to the type `OrganizationType`. It constitutes the first parameter in the ATSM document. `OrganizationType` is again composed by a number of elements in a sequence. The elements `has-sub-unit` and `has-size` point to simple types, which in turn each result in an ATSM parameter (namely, `Organization-Unit` and `OrganizationOrganization-Size`). The element `organization-part-of` points to `OrganizationType` and thus constitutes a loop, which results in the recursive process terminating at this point. Yet, `OrganizationType` is added as parameter once again. The elements `affiliated-person` and `headed-by`, however, point to the complex type `Affiliated-Person`, resulting in two additional ATSM parameters. Again, `Affiliated-Person` is recursively processed. Its only element, `has-affiliation`, points to the complex type `OrganizationType`. Thus, two more parameters are generated. Afterwards, the process terminates due to a loop being detected.

Overall, the following ATSM parameters are generated (the number in braces specifies the respective recursive depth in the generation process):

- OrganizationType (1)
- Organization-Unit (2)
- OrganizationType (2)

- Affiliated-Person (2)
- OrganizationType (3)
- Affiliated-Person (2)
- OrganizationType (3)
- OrganizationOrganization-Size (2)

In the case of OWL-S, a comparable process is followed. Each concept that is defined by a `parameterType` in OWL-S is recursively resolved and directly results in an ATSM parameter. Then, all applicable object properties are determined for that concept, and each concept in the property's range is traversed, again leading to additional parameters. The process continues up to a specified recursive depth or until no more properties are applicable to a concept.

Listing A.4: Example ATSM Document (Excerpt)

```
@prefix tns:     <http://dmas.dfki.de/axis/services/SurfingorganizationDestination#> .
@prefix atsm:    <http://kom.tu-darmstadt.de/ns/atsm#> .

<tns:atsm.service(surfingorganization_destination_service)>
      a       atsm:Service ;
      atsm:hasInterface <tns:atsm.interface(surfingorganization_destination_service/
           SurfingorganizationDestinationSoap)> ;
      atsm:hasName "surfingorganization_destination_service" .

<tns:atsm.interface(surfingorganization_destination_service/SurfingorganizationDestinationSoap)>
      a       atsm:Interface ;
      atsm:hasOperation <tns:atsm.operation(surfingorganization_destination_service/
           SurfingorganizationDestinationSoap/get_DESTINATION)> .

<tns:atsm.operation(surfingorganization_destination_service/SurfingorganizationDestinationSoap/get_DESTINATION
     )>
      a       atsm:Operation ;
      atsm:hasInput <tns:atsm.messageReference(surfingorganization_destination_service/
           SurfingorganizationDestinationSoap/get_DESTINATION/In)> ;
      atsm:hasName "{http://dmas.dfki.de/axis/services/SurfingorganizationDestination}get_DESTINATION" ;
      atsm:hasOutput <tns:atsm.messageReference(surfingorganization_destination_service/
           SurfingorganizationDestinationSoap/get_DESTINATION/Out)> ;
      atsm:hasPattern <http://www.w3.org/ns/wsdl/in-out> .

<tns:atsm.messageReference(surfingorganization_destination_service/SurfingorganizationDestinationSoap/
     get_DESTINATION/In)>
      a       atsm:MessageReference ;
      atsm:hasName "In" ;
      atsm:hasParameter <tns:atsm.parameter(surfingorganization_destination_service/
           SurfingorganizationDestinationSoap/get_DESTINATION/In/OrganizationType)> [..] .

<tns:atsm.messageReference(surfingorganization_destination_service/SurfingorganizationDestinationSoap/
     get_DESTINATION/Out)>
      a       atsm:MessageReference ;
      atsm:hasName "Out" ;
      atsm:hasParameter <tns:atsm.parameter(surfingorganization_destination_service/
           SurfingorganizationDestinationSoap/get_DESTINATION/Out/DestinationType)> .

<tns:atsm.parameter(surfingorganization_destination_service/SurfingorganizationDestinationSoap/get_DESTINATION
     /In/OrganizationType)>
      a       atsm:Parameter ;
      atsm:hasName "{http://dmas.dfki.de/axis/services/SurfingorganizationDestination}OrganizationType" ;
      atsm:hasSemanticReference
              <http://127.0.0.1/ontology/portal.owl#Organization> .

<tns:atsm.parameter(surfingorganization_destination_service/SurfingorganizationDestinationSoap/get_DESTINATION
     /Out/DestinationType)>
      a       atsm:Parameter ;
      atsm:hasName "{http://dmas.dfki.de/axis/services/SurfingorganizationDestination}DestinationType" ;
      atsm:hasSemanticReference
              <http://127.0.0.1/ontology/travel.owl#Destination> ;
      atsm:hasSyntacticType
              "{http://www.w3.org/2001/XMLSchema}string" .

[..]
```

A.2 Service Registry Standards

In the following, the both service registry respectively repository standards applied in Chapter 4 will be presented, i.e., UDDI (Section A.2.1) and ebXML (Section A.2.2).

A.2.1 Universal Description, Discovery and Integration

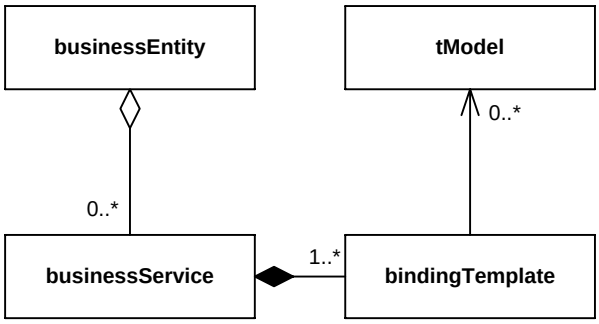


Figure A.3: The UDDI (Version 3.0) Core Data Model (taken from [62])

Universal Description, Discovery and Integration (UDDI) is an Organization for the Advancement of Structured Information Standards (OASIS) standard with Version 3.0 being the most recently released specification [62]. The full name of this registry standard already indicates that UDDI is not restricted to Web services, but represents a "universal" concept, so that every content, which relies on XML, can be published in the registry. However, UDDI has gained most attention as a framework to describe, publish, search for (discover), and finally integrate Web services into an arbitrary software system.

A UDDI registry is made up from the following categories [62]:

White Pages provide additional information about enterprises (e.g., address, contact, known identifiers). This information can then be queried by a service requester in order to decide, whether he wants to use a service provided by the respective company.

Yellow Pages provide a classification scheme according to industry sectors, so that a service requester is able to retrieve the services of all the service providers from a specific line of business.

Green Pages are applied to publish descriptions for each service. They allow manual browsing in case the service provider and its industry sector are both unknown and are linked with the technical data, which provides information for the utilization of a Web service in a machine-processable manner.

The data model of UDDI is based on XSD and consists of four core entity types (cp. Figure A.3) [62]:

`businessEntity` instances represent companies or organizations and contain the name of the company or organization as well as data of contact persons. Each `businessEntity` also provides information about the offered services by its contained `businessService` instances.

`businessService` instances model a service offer and represents an abstract description of either a group of concrete services constituting the offer or a single service providing different access points. A `businessService` instance may contain one or more `bindingTemplate` instances, each describing a corresponding Web service. Please note that in UDDI Version 2.0, it is not mandatory to define a `bindingTemplate` – however, there is no practical impact of this difference regarding the work presented in Section 4.3.

`bindingTemplate` entities are used to store the technical information, which is required for the invocation of a concrete Web service, e.g., the network address in terms of a URL. In addition, a `bindingTemplate` contains references to `tModels` to describe further technical details.

`tModel` represent a unique concept or construct, e.g., the contract of a Web service in form of a WSDL file or classification information. They are intended to describe compliance with shared standards, concepts or specifications. Since each `tModel` is holding a unique key, tModels can be reused within the registry. The technical documents a `tModel` embodies are not stored in the registry itself. Instead, a `tModel` stores metadata about the documents and contains the address, where a document can be found.

In the context of the implementation presented in Section 4.3, tModels play an important role. As it can be seen from Figure A.4, tModels are made up from a unique key, a name, an optional number of descriptions, an optional `overviewDoc` element, an optional `identifierBag`, and `categoryBag` objects. The `overviewDoc` is made up from an optional number of descriptions as well as an optional `overviewURL`. There are two kinds of tModels:

Specification tModels are used in order to describe a service type definition. If an object is published in the UDDI, it may reference a specification tModel in order to specify its compliance with a type definition.

Category systems and identifier systems tModels enable the categorization and identification of objects published in the UDDI.

The following differences between UDDI 2.0 and 3.0 affect the work at hand [62]:

- Instead of UUID-based keys, URI-based keys are applied.
- New element `keyedReferenceGroup`.
- Elements are allowed to possess several `overviewDocs`.
- `keyedReferences` need to possess a `tModelKey`.

If a WSDL-based service is published in UDDI, it is represented by a tModel whose `overviewURL` points to the actual WSDL-URL; the `categoryBag` points to a tModel which specifies a Web service described in WSDL. Hence, the `categoryBag` is used to categorize and specify that this tModel is offering a WSDL. Therefore, `keyedReferences` are applied: In UDDI, `keyedReferences` are pairs of names and values, which represent a variable `keyName` and its value `keyValue`. Each `keyedReference` features a `tModelKey` referencing a `tModel`, which defines the type of the variable.

For interacting purposes, UDDI implementations provide several APIs, from which the *Inquiry* and *Publication* APIs are mandatory components. The *Publication* API can be utilized to register content, respectively services, and the *Inquiry* API can be used to search the registry. Concerning the query mechanisms of UDDI, an exact or approximate string matching can be performed and the adherence to tModel instances can be verified [62]. Possible extensions for the integration of other query formalisms are presented in Section 4.5.1.

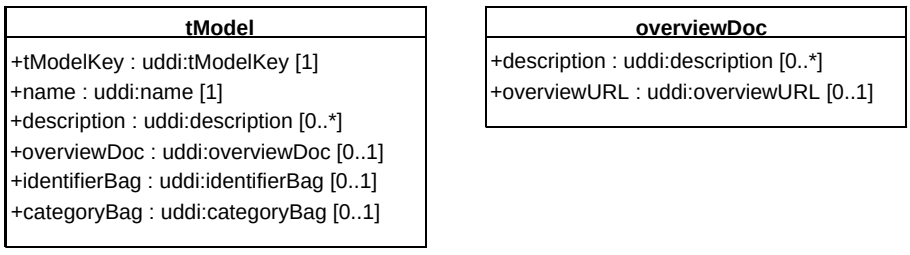


Figure A.4: Data Type tModel

A.2.2 Electronic Business using Extensible Markup Language

ebXML is a family of OASIS standards and specifications which are partly also standardized by the International Organization for Standardization (ISO). Generally, ebXML provides a modular suite of specifications for enterprises to perform business over the Internet (e.g., message exchange, registration of services). The project incorporates five layers of XML-based specifications [196]:

Messaging Service defines a standard method for secure and reliable message exchange between enterprises, which is independent from the utilized communication protocol.

Collaboration Protocol Profile and Agreements specify the (shared) technical capabilities and preferences of the collaborating parties.

ebXML Business Process Specification Schema provides a standard language to define and configure multi-party business collaborations.

Registry-Repository provides capabilities to register and retrieve information resources in distributed systems.

Core Components describe a methodology based on semantic building blocks to define the type of business data in a commonly understandable way.

Within the work at hand, only the specifications for registries and repositories are relevant, and thus will be further discussed in the following. In this context, two documents are currently available in Version 3.0 as approved OASIS standards: the ebXML Registry Information Model (RIM) and the ebXML Registry Services and Protocols (RS) [85, 86]. The former specifies the underlying data model of the registry and the latter describes the services provided by the registry and the protocols used for interacting with it.

The ebXML Registry terminology distinguishes between the terms “repository” and “registry”. A repository is used to store electronic content of any type (e.g., XML documents, images); the associated metadata describing the content is managed within a registry. The terms “repository item” and “registry item” are used analogously. An ebXML Registry covers both, a metadata registry and a content repository. Typically, a database is used as persistent store for metadata and content. Although the registry and the repository are considered as distinct components from an architectural point of view, they are both accessed by the interfaces provided by the ebXML Registry [85, 86]. As stated in Section 2.1.3, in this thesis the term “registry” is used to refer to both, registry and repository, unless indicated otherwise.

The classes which are used to represent the metadata of the objects stored in the registry and their relationships are defined in the ebXML RIM [85]. An ebXML Registry may further implement different profiles, each defining a processing standard as well as extensions and restrictions of the core ebXML features for a specific type of content. The *ebXML Registry profile for Web Services* defines the publication, management and discovery of Web service artifacts [191]. In general, most metadata classes are subclasses of the `RegistryObject` base class within the RIM [85]. The RIM classes which are relevant for the registration of Web services in an ebXML Registry implementing the Web service profile are depicted in Figure A.5. Within the RIM, a service is represented by an instance of `RegistryObject`’s subclass `Service`. An instance of the `Service` class contains one or more instances of the `ServiceBinding` class, which provide technical information on how to access a concrete service instance.

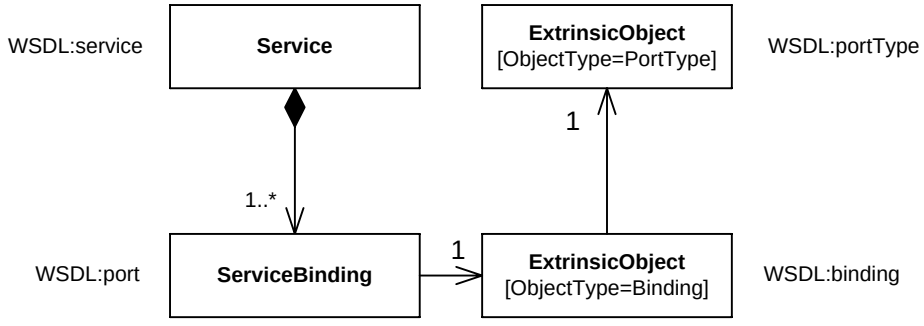


Figure A.5: Mapping of WSDL information to ebXML RIM (taken from [191])

Generally, classification schemes are used in ebXML to classify `RegistryObject` instances [85]. In doing so, the type of content, which has been submitted to the registry as an instance of the `ExtrinsicObject` class, can be classified using nodes of the classification scheme `ObjectType`. WSDL files are stored in the registry as `ExtrinsicObjects` and classified with the canonical WSDL classification node [85]. When submitting a WSDL document to the registry, a cataloging service is invoked, which performs a mapping of the WSDL components to the ebXML RIM. During this process, the `service` and `port` sections are represented by the respective RIM classes, whereas the `binding` and `portType` sections are stored as extrinsic objects (cp. Figure A.5). All components are classified by the designated `ObjectType` specified in the ebXML Web service profile, i.e., with subnodes of the canonical WSDL classification node. This is due to the fact, that the default service information model as part of the RIM also supports the registration of other types of services than Web services [85], hence represents a generic service model. Consequently, the components which are specific to a certain kind of service have to be stored as extrinsic objects and classified using custom-built classification schemes. The ebXML Registry profile for Web Services is only defined for WSDL 1.1 documents.

Concerning the query capabilities of ebXML, a flexible and extensible query mechanism is provided, so that an implementation may introduce additional queries. Search criteria also propagate upon attributes of lower level objects as part of higher level objects to be discovered. The WSDL service discovery query specified in the ebXML Registry profile for Web Services allows the discovery of service instances using an arbitrary number of parameters, which refer to the attributes of the objects contained in the WSDL component model (e.g., `$name of the service`, `$port.accessURI`, `$binding.protocolType`). A parameter value typically contains a string pattern including wildcard characters "%" to be matched against the respective attribute value [191].

Clients submitting a query to the registry may use one of the supported query syntaxes. Generally, each ebXML registry must support an XML-based query syntax called "Filter Query syntax". When a filter query is submitted to the registry, the `RegistryObjects` are considered as a virtual XML document tree and pruned or filtered according to the specified query parameters. All remaining objects matching the filter expressions are returned as a collection to the client. In addition, an ebXML Registry implementation may support SQL as query syntax [86]. Finally, a GUI is recommended for ebXML registries in order to facilitate the creation of queries providing predefined fields for each prospective query parameter [191].

A.3 The Resource Description Framework and the Web Ontology Language

RDFS and OWL are two major standards for the representation of ontologies which have been standardized by W3C. Most recently, W3C has published OWL 2 Web Ontology Language (OWL 2) which is an extension of the original OWL specification [36, 108, 182]. In the following, RDF, RDFS, and OWL will be introduced with respect to the aspects required in this thesis.

A.3.1 Resource Description Framework

RDF represents a framework to express information, i.e., metadata, about resources on the Web and is available as a set of six W3C recommendations. RDF is not restricted to retrievable Web resources only, but also allows for the description of arbitrary resources in general. Basically, RDF is intended for automated processing by applications in order to facilitate the exchange of information [177].

RDF is based on the definition of *triples*. A triple defines a relationship between a *subject* and an *object* using a *predicate* [177]:

Subject represents a resource described by a statement and is identified by a URI.

Predicate represents a specific property of the subject. A predicate is also a resource that can be described by a statement.

Object represents the value of the predicate and is either a resource or a literal.

A simple example of RDF in N3 serialization, taken from the *RDF Primer* [177], is depicted in Listing A.5. RDF also provides an XML serialization called RDF/XML [16].

Listing A.5: RDF Code in N3 Serialization

```
<http://www.example.org/index.html>
  <http://purl.org/dc/elements/1.1/creator>
    <http://www.example.org/staffid/85740> .
<http://www.example.org/index.html>
  <http://www.example.org/terms/creation-date>
    ''August 16, 1999'' .
<http://www.example.org/index.html>
  <http://purl.org/dc/elements/1.1/language>
    ''en'' .
```

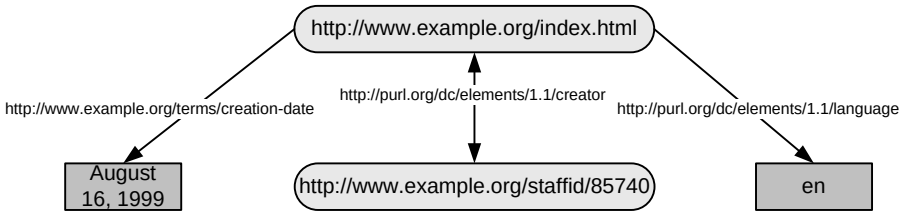


Figure A.6: Visualization of RDF Code as Graph (taken from [177])

Due to the underlying structure, RDF statements can be modeled as graphs, where the nodes either represent subjects or objects, i.e., resources or literals, and the arcs are labeled with the name of a predicate, i.e., indicating a specific property. A sample graph is depicted in Figure A.6 and expresses the shortened English statements shown in Listing A.5.

Because of its generic character, RDF provides insufficient instruments for building complex ontologies. Most notably, it does not allow to impose restrictions regarding the validity of statements. For instance, in the case of an English language ontology, we would probably like to restrict subjects and objects to the class of words. Accordingly, possible predicates are derived from a predefined set of relationships between words, such as synonym and antonym.

Based on RDF, RDFS defines a more expressive language [36]. It introduces the concept of *classes* and *properties*. Latter allow to define specific relations between objects. In the course of this thesis, *subsumption relations* between classes are of elevated interest. These are relations where classes constitute either a generalization or specialization of other classes, which is a fundamental relation in ontologies.

For instance, related to the domain of biology, “animal” could be the root class, with the subclass of “mammal” being a specialization of it. Again, “cat” and “dog” could be subclasses of “mammal”. Accordingly, the more generic concept “mammal” subsumes the classes “cat” and “dog”. As a subsumption relationship is transitive, “animal” also subsumes “cat” and “dog”.

A more formal introduction to subsumption relationships is given in Section A.3.3.

A.3.2 Web Ontology Language

Because RDFS also suffers from a number of limitations (e.g., it is not possible to define negative assertions), the W3C has devised OWL as a more powerful standard for knowledge representation. OWL makes use of the representation of information about resources from RDF and the declaration of classes and properties in RDFS [36, 177]. With OWL, it is possible to define logical combinations (intersections, unions, or complements) of classes, i.e., in order to build complex classes from atomic classes. OWL also facilitates the declaration of individuals (objects) and their specific property values. It is possible to declare transitive, symmetric, or functional properties. Furthermore, it is possible to define equivalence between classes and properties and disjointness between classes. Possible relationships for individuals are equality and inequality [110]. The most important extension of OWL (compared to RDFS) is the possibility to restrict properties regarding values and cardinalities [110]. Value constraints put restrictions on the range of the property, if applied to a particular class description. Cardinality constraints put restrictions on the number of values a property can adopt, if applied to a particular class description [67].

OWL provides three increasingly expressive sublanguages, which offer different levels of expressiveness and complexity, but do all make use of RDF as their syntax [182]:

OWL Lite represents the simplest version of OWL and allows to define a classification hierarchy and simple constraints.

OWL DL exhibits maximum expressive power while providing computational guarantees, i.e., computation of conclusions in finite time.

OWL Full offers maximum expressiveness and allows to extend the meaning of the predefined RDF/OWL vocabulary, thus provides no computational guarantees.

In the context of this thesis, OWL DL is used as knowledge representation language. OWL DL provides a syntax for describing ontologies and is very close to the DL language $SHOIN(\mathbf{D})$ [110]. As Horrocks et al. state, "OWL DL [...] can be viewed as expressive Description Logics, with an ontology being equivalent to a Description Logic knowledge base" [112]. Classes from OWL correspond to concepts as defined for DL, properties correspond to roles, and individuals correspond to objects. A brief introduction into DL can be found in the next section.

OWL provides the means to *define* ontologies. The automatic inference of relations requires an appropriate software component, called a *reasoner*. It allows loading and parsing ontologies and consecutively processing them, based on logical reasoning. I.e., ontologies as such are of limited use without a corresponding reasoner that permits to infer implicit knowledge. In this thesis, we make use of *Pellet*, a OWL DL reasoner applied by many researchers in the Semantic Web community [230].

A.3.3 Description Logics

Description Logics (DL) is used as a formalism to represent the knowledge about a particular application domain [192]. In this context, DL is a family of knowledge representation formalisms used to model concepts, roles and individuals, and the corresponding relationships [11].

DL is based on the definition of concepts which constitute the *terminology* of a domain ("TBox"), i.e., a vocabulary of a domain. Concepts denote sets of individuals, while *roles* denote binary relationships between individuals [11]. Per se, concepts are atomic, however, DL allows to build complex *descriptions* of concepts and roles. With the TBox in place, it is possible to use the vocabulary in order to describe properties and objects and individuals in the domain, i.e., define *assertions* about named individuals. This "ABox" describes a specific state in terms of concepts and roles by a finite set of assertions [11]. Together, TBox and ABox form a so-called *knowledge base*. It should be noted, that there are different viewpoints on the last-mentioned aspect. Mädche et al., e.g., define the knowledge base as the ABox, while an ontology would be the corresponding TBox [173].

DL-based languages have got formal, logic-based semantics and allow reasoning, i.e., to *infer* implicit knowledge from the explicitly defined knowledge. In the context of this thesis, the determination of subconcepts and superconcepts, called *subsumption* relationships play an important role. The basic idea is that if a concept D is more general than a concept C, it subsumes C: $C \sqsubseteq D$. Using subsumption reasoning, it is possible to derive if an object (individual) is an instance of a given concept. Another reasoning task is the determination if a description is *satisfiable*, i.e., non-contradictory [11].

A DL language is based on syntax rules that define which expressions are legal; semantics determine the meaning. *Constructors* define how concept terms can be formed – examples are *equivalent*, *disjointness*, and *covering*. Constructors can be used in order to define *terminological axioms*, i.e., statements about how concepts and roles are related to each other. If two subclasses/subconcepts are disjoint, an individual (like an input referencing a semantic concept in SAWSDL) cannot belong to both subclasses. In an ontology, this is modeled through negation. Covering is used in order to define if a set of subclasses fully covers their superclass, i.e., the superclass is the union of a set of mutually disjoint subclassed [34].

For a broader presentation of DL, we refer to *The Description Logic Handbook* [12].

Listing A.6: Example SPARQL Query

```
1  PREFIX dc: <http://purl.org/dc/elements/1.1/>
2  PREFIX ex: <http://www.example.org/>
3  PREFIX exterms: <http://www.example.org/terms/>
4
5  SELECT ?name
6  WHERE
7  {
8   ex:index.html  dc:creator    ?staffid .
9   ?staffid       exterms:name  ?name .
10  FILTER regex(?name, "Smith")
11 }
12 ORDER BY ASC(?name)
```

SPARQL is a query language for RDF standardized by the W3C [215]. It provides an SQL-alike syntax and returns result sets in form of tables or RDF graphs. An example of a SPARQL query defined for an extended version of the sample RDF graph shown in Figure A.6 is depicted in Listing A.6. It is assumed, that the Web page described in the extended RDF graph has been constructed by a number of creators, each specified in the graph in the same way as "John Smith". The example retrieves the names of all creators in ascending order, who developed the Web page `ex:index.html` and whose name contains "Smith".

Basically, the SPARQL query language consists of four components [118, 215]:

RDF Dataset denotes one or more RDF graphs in an RDF data store. One graph represents the *default* graph, while the other graphs are *named* graphs. SPARQL queries are executed against an RDF dataset and may refer to one or more graphs.

Graph Patterns represent sets of triple patterns. As it can be seen in Listing A.6 (Lines 8–9), the query statements are specified in terms of triple patterns replacing one or more of the elements by variables. In addition, SPARQL defines several special types of graph patterns, so that graph patterns can be grouped, and optional and alternative graph patterns can be specified. Finally, value constraints in terms of `FILTER`s can be specified, which make use of predefined filter functions. In the example query above, the values of the `?name` variable must contain the substring "Smith" (Line 10).

Solution Modifiers alter the sequence of a result set. By default, a SPARQL query returns an unordered sequence of solutions. For example, the order can be specified, the number of results can be limited and also restricted to unique solutions.

Query Forms are also provided in SPARQL:

- The `SELECT` form simply returns the variables specified in the query.
- Using the `CONSTRUCT` form, a single graph is returned, which consists of the triples represented by the query statements, in which the variables are replaced by the solutions.
- In order to test for the existence of a solution for a given graph pattern, the `ASK` query can be utilized.

A sample RDF Dataset is depicted in Listing A.7, a corresponding SPARQL query can be found in Listing A.8 (both examples are taken from [215]). In the example, the default graph stores the provenance information of the named graphs, while the actual RDF data is stored within the named graphs identified by different IRIs [215]. The corresponding query result contains the date of the graph, where the query pattern matched, and the name of the person described within the respective graph.

Apart from its genuine application area, i.e., being a query language for RDF, there are several extensions of SPARQL. These integrate further processing possibilities into the query language or facilitate the usage of SPARQL in different settings [39, 125].

Listing A.7: A Sample RDF Dataset

```
# Default graph
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix g:  <tag:example.org,2005-06-06:> .

g:graph1 dc:date ''2004-12-06'' .
g:graph2 dc:date ''2005-01-10'' .

# Graph: locally allocated IRI: tag:example.org,2005-06-06:graph1
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

_:a foaf:name ''Bob'' .

# Graph: locally allocated IRI: tag:example.org,2005-06-06:graph2
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

_:b foaf:name ''Alice'' .
```

Listing A.8: A Sample SPARQL Query Ranging over Different Named Graphs

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc:   <http://purl.org/dc/elements/1.1/>

SELECT ?name ?date
WHERE
{  ?g dc:date ?date .
   GRAPH ?g
   { ?person foaf:name ?name }
}
```

B Evaluation Setup

While the query formulation approaches presented in Chapter 4 are evaluated using the qualitative attributes and requirements defined in Section 4.2, the evaluation of the matchmakers presented in Chapter 3 aims on effectiveness and (to a lesser extent) efficiency. In the following, the metrics, data sets, and frameworks used in this evaluation will be presented. Summarized, the matchmaking approaches are evaluated following the *Cranfield paradigm*, which is a well-known paradigm for the evaluation of IR algorithms [254]. The evaluation focuses on quantitative terms while also incorporating qualitative aspects, which will be presented in Section B.1. The evaluation modality in terms of evaluation environment and test data set are presented in Section B.2. The applied quantitative metrics from the field of IR are defined in Section B.3. Finally, cross-validation is regarded in Section B.4.

B.1 Evaluation Approach

In their work on the evaluation of semantic service discovery, Küster et al. state that a thorough evaluation of semantic service discovery approaches should answer at least the following questions regarding the assumptions that affect the evaluation and the dimensions measured [157]. As long as not stated otherwise, the following information applies to all matchmaking approaches presented in Sections 3.2 and 3.3 as well as the query formalisms presented in Sections 4.3 and 4.4.

Assumptions of the evaluation:

- *Determination of the formalisms and logical language used*: In the work at hand, we make use of SAWSDL (based on WSDL 2.0) as Web service standard [32, 79]. SAWSDL-based service descriptions are mapped to ATSM in order to avoid the costly (in terms of runtime performance) processing of SAWSDL-based services. ATSM also includes a mapping from OWL-S (cp. Appendix A.1.3), however, this mapping has not been regarded in the matchmaking evaluation. SAWSDL and ATSM do not restrict the type of semantic concepts a semantic annotation should reference. However, in this thesis it is assumed that the semantic concepts are formally defined in an OWL DL-based ontology (cp. Section 2.2.3 and Appendix A.3.2) [182]. Regarding the matchmaking approaches, service requests need to be formulated in SAWSDL (cp. description of SAWSDL-TC below), in order to facilitate comparability with the other matchmaking approaches. The format of queries and service repositories is defined in the corresponding Sections 4.3 and 4.4.

- *Scope for discovery*: While the discovery approaches presented in this thesis (both matchmakers and query languages) could be applied in a different context, we assume that the service requester is able to adapt either his own software or map components (e.g., inputs and outputs) of the service offers. Of course, this does not apply to services that exactly meet the requirements of the requester. In order to facilitate the automatic invocation and composition of the service result set, the result sets could be limited to only exact matches. However, this would certainly worsen the overall recall value for a query and is therefore not applied in the matchmaking evaluation.

- *Expected outcome of discovery*: The expected outcome of the discovery process is a sorted list of services, starting with the best fitting service to a particular request. The result sets are afterwards quantitatively evaluated using IR metrics like precision and recall. If two or more service offers share the same similarity value, we randomize the order of these specific service offers in the result set.

Dimensions measured:

- *Runtime performance*: The runtime performance in terms of AQRT is evaluated for all matchmaking approaches. However, the runtime performance might depend on (automatic) activities that need to be carried out prior to the actual query execution, especially the caching of similarity values.

- *Scalability*, in terms of the number of services, has not been in the focus of our approaches and is therefore not examined.
- *Complexity of service descriptions*: In order to generate sound matchmaking results, we anticipate that service descriptions feature semantic annotations at least in the service signature, i.e., that semantic type declarations are given for message inputs and outputs. If those are missing or cannot be processed (for any service component), a syntax-based fallback strategy is applied (cp. Section 3.2.2).
- *Guarantees provided by a match*: In contrast to the related work by, e.g., Paolucci et al. or Klusch et al., we do not guarantee a match to provide a certain lower bound of matching like, e.g., a certain degree of subsumption matching [140, 201]. Instead, a high numerical matching value indicates that a service offer better fulfills the requirements defined by the requester than a lower numerical value.
- *Standard compliance and reusability*: The matchmaking approaches presented in this thesis can be reused as long as they are applied to SAWSDL (based on WSDL 2.0) and semantic concepts are denoted in OWL DL. Plugins for SME2 (see below) have been developed which make it possible for fellow researchers to make use of the matchmakers and compare them with their own approaches.

The focus of the matchmaking evaluation is the *relative* performance of the matchmakers presented in this thesis in comparison to the approaches presented by other research teams. According to Voorhees, for such a comparison, an evaluation of IR metrics following the *Cranfield paradigm* is suitable [253]. The Cranfield paradigm is based on test collections which feature a set of documents (here: semantically described Web services), a set of information needs (here: a set of queries), and a set of relevance judgments (here: a set of optimal result sets for every query). Based on this, it is possible to define the *precision*, *recall*, and constitutive metrics of a particular approach.

Both matchmakers presented in this thesis offer parameters, which allow to evaluate different versions and variants of LOG4SWS.KOM and COV4SWS.KOM. As the evaluation shows, we were able to identify which aspects of our matchmakers lead to the observed results. Apart from the evaluation of IR metrics, we also discuss the matchmaking approaches in terms of qualitative aspects.

B.2 Evaluation Environment

The evaluation environment for the matchmaking approaches presented in Chapter 3 is made up from two parts: The actual evaluation framework SME2 (Section B.2.1) and the SAWSDL-TC test data set (Sections B.2.2 and B.2.3).

B.2.1 SME2 Framework

For the evaluation of our matchmaking approaches, we have used the SME2[1] framework (Version 2.1). SME2 is also applied in the S3 Contest[2], which aims at the retrieval performance evaluation of matchmakers for SWS [142]. Thus, it was possible to compare the results of LOG4SWS.KOM and COV4SWS.KOM with the evaluation results of state-of-the-art matchmakers for SAWSDL.

The SME2 framework consists of a GUI-based tool for the evaluation of Web service matchmakers and libraries that facilitate the integration of matchmakers into the framework. The GUI can be utilized in conjunction with OWL-S and SAWSDL test collections (see below). It automatically calculates a number of numerical result quality measures. Most importantly, this includes the "classic" IR metrics *recall* and *precision* (cp. Section B.3). Additionally, the tool computes the AP, which corresponds to the mean precision rate over all recall levels. The statistical figures are completed by performance measures, such as the Overall and Average Query Response Time. Furthermore, a Friedman test is provided and can be used to evaluate if the differences between evaluation results are statistically significant. The threshold value for p, which indicates the level of significance (or probability of error) is 0.05.

[1] http://projects.semwebcentral.org/projects/sme2/
[2] http://www-ags.dfki.uni-sb.de/~klusch/s3/

SME2's evaluation for SAWSDL is based on the notion of binary relevance. I.e., the framework assumes that for a given service request, a set of relevant (and accordingly, irrelevant) service offers can be identified. Relevant, in that sense, means that the service offer does not completely fail to satisfy the service request. SME2 expects a ranked result set. Ideally, it should solely consist of relevant services, or, at least, all relevant services should be ranked very high. The specific similarity of each service offer with the service request is not taken into account. However, in the most recent versions of SME2, *graded relevance* is reckoned, too. As the name implies, in contrast to binary relevance, graded relevance incorporates different degrees of relevance. So, instead of expecting that service offers either completely fail or are a perfect match to a service request, different, finer-grained grades like "possible match" or "partial match" are defined in the relevance sets [152, 154, 246]. Unfortunately, there is currently no corresponding, generally accepted test data collection for SAWSDL that could have been applied in order to evaluate the matchmakers at hand with respect to graded relevance (cp. the description of SAWSDL-TC in the following section).

B.2.2 SAWSDL-TC Test Collection

To the best of our knowledge, the test data collections applied in the S3 Contest are today the largest and most-accepted sets of SWS applied for testing and evaluating algorithms for SWS discovery and matchmaking. In the S3 Contest, two test collections from `www.semwebcentral.org` are used – with *OWLS-TC version 3.0 Revision 1* and *SAWSDL-TC1* as the current versions of service retrieval test collections for OWL-S and SAWSDL [122]. As SAWSDL is the SWS standard applied in this thesis, we apply SAWSDL-TC as test collection.

SAWSDL-TC was released in 2008 and consists of 894 semantically annotated WSDL 1.1-based Web services, which cover differing domains from education and medical care to food and travel, etc. The set contains 26 queries and a relevance set for each query is provided which can be used in order to compute IR measures. The queries are defined as "queries by example" (cp. Section 4.1). For the evaluation of matchmaking approaches, we apply these queries directly. For the evaluation of the query languages, we have mapped the original queries as described in Chapter 4. The test suite is completed by 24 ontologies, which are referenced using semantic annotations of services.

As SAWSDL-TC is WSDL 1.1-based, it was necessary to convert the test collection to WSDL 2.0, which is the designated service format for the matchmakers presented in this thesis. This is done by the XSLT-stylesheet for converting WSDL 1.1 to 2.0 introduced in Appendix A.1.2. As no new semantic annotations have been added to or eliminated from the service descriptions, and the structure of service descriptions is eventually the same, it is possible to compare the performance of the matchmakers at hand with those of other matchmakers which make use of SAWSDL-TC.

While SAWSDL-TC is a de facto standard in the evaluation of Web service matchmakers, it suffers from a number of limitations. First of all, the scope of SAWSDL-TC is clearly on the evaluation of IR measures; other desirable dimensions of evaluation as, e.g., the incorporation of different viewpoints what a valid result to a service request might be [153], are not regarded. Second, from a more technical point of view, Web services in SAWSDL-TC are solely annotated at the parameter level. Interface and operation components are not annotated at all. Generally, each service solely contains one interface with exactly one operation. That way, more sophisticated matchmaking approaches on levels beyond the parameters cannot be fully evaluated. Furthermore, some obvious inconsistencies can be found in the compilation of relevance sets. E.g., at least one pair of services that has an identical set of parameters and semantic annotations is not contained in the same relevance set. Furthermore, six services that are contained in the relevance sets are not included in the directory of service offers, which leads to a slight reduction in recall for the corresponding requests. Yet, due to the lack of a more suitable test collection and for the ease of comparison, we have opted to utilize the WSDL 2.0-based version of SAWSDL-TC in the course of our evaluation. As the SAWSDL-TC service collection is completely independent from the work at hand, we meet the requirement of "fair testing" as we do not make use of an artificially created test data set, which meets the requirements of the matchmakers presented in this thesis [156, 261].

An overview of the queries and relevance sets of SAWSDL-TC is provided in the next section.

B.2.3 SAWSDL-TC: Queries and Relevance Sets (Overview)

Table B.1 shows the queries from SAWSDL-TC and the corresponding number of services from the respective result set. Size is given in Kbyte.

Table B.1: SAWSDL-TC: Queries and Relevance Sets (Overview)

No.	Size	Query	Number of Services in Relevance Set
1.	5	book_price_service	37
2.	5	bookpersoncreditcardaccount__service	16
3.	6	bookpersoncreditcardaccount_price_service	21
4.	4	car_price_service	40
5.	6	citycountry_hotel_service	23
6.	5	country_skilledoccupation_service	74
7.	5	dvdplayermp3player_price_service	14
8.	5	geographical-regiongeographical-region_map_service	15
9.	5	geopolitical-entity_weatherprocess_service	23
10.	5	governmentdegree_scholarship_service	40
11.	5	governmentmissile_funding_service	37
12.	4	grocerystore_food_service	27
13.	4	hospital_investigating_service	19
14.	4	maxprice_cola_service	13
15.	4	novel_author_service	22
16.	4	preparedfood_price_service	25
17.	5	recommendedprice_coffeewhiskey_service	18
18.	7	researcher-in-academia_address_service	16
19.	7	shoppingmall_cameraprice_service	18
20.	4	surfing_destination_service	32
21.	4	surfinghiking_destination_service	39
22.	7	surfingorganization_destination_service	13
23.	4	title_comedyfilm_service	14
24.	4	title_videomedia_service	12
25.	5	university_lecturer-in-academia_service	20
26.	5	userscience-fiction-novel_price_service	28

The queries and relevance sets from the SAWSDL-TC test data set shown in Table B.2 were utilized in order to manually determine optimal numerical equivalents for the DoM levels presented in Variants B and C of LOG4SWS.KOM (cp. Section 3.4.1).

The optimal numerical equivalents have been determined by testing different combinations of weightings. As it is the case in Section 3.4, the numerical equivalents for exact and fail matches have been set to 1 and 0 respectively. The numerical equivalents for super and sub matches have been given weightings from 0 to 1 in 0.1-steps. The best evaluations results have been observed for $\{1, 0.8, 0.6, 0\}$.

Table B.2: Service Collection for the Determination of Numerical DoM Equivalents in LOG4SWS.KOM

No.	Query	Number of Services in Relevance Set
4.	car_price_service.wsdl	40
5.	citycountry_hotel_service.wsdl	23
11.	governmentmissile_funding_service.wsdl	37
15.	novel_author_service.wsdl	22
16.	preparedfood_price_service.wsdl	25
19.	shoppingmall_cameraprice_service.wsdl	18
20.	surfing_destination_service.wsdl	32
23.	title_comedyfilm_service.wsdl	14
24.	title_videomedia_service.wsdl	12
25.	university_lecturer-in-academia_service.wsdl	20
Total	10	243

B.3 Information Retrieval Performance Measures

Information retrieval refers to the automated retrieval of documents in general. In the context of this thesis, we will restrict the term to the specialized domain of *ad hoc IR*, where information is retrieved in response to a specific query. As Grossman and Frieder put it, "Information Retrieval is devoted to finding relevant documents, not simple matching to patterns" [93].

Thus, IR comes into play when a set of documents, which is deemed to be relevant to a specified query, is to be retrieved from a potentially very large collection of documents. Generally, this also includes the ranking of retrieved documents in terms of relevance, compared to a query. Probably the most prominent use of IR nowadays is in the field of Internet search engines. In accordance with searching on the Internet, Web service matchmaking or service discovery is also an application of IR [157]. The search space is constituted by the Web services registered in a service registry, whereas the query is, for instance, given by a query by example. It seems reasonable to apply "classic" IR evaluation metrics like *precision* and *recall* which are defined as follows [14, 176]:

If A is the set of all relevant documents for a request and B is the set of all retrieved documents for a request, then we can make use of the following metrics:

- $Recall = \frac{|A \cap B|}{|A|}$, i.e., the fraction of *relevant* documents that are *retrieved*.
- $Precision = \frac{|A \cap B|}{|B|}$, i.e., the fraction of *retrieved* documents that are *relevant*.

These metrics indicate shares, i.e., their values can range from 0 to 1. Ideally, a retrieval algorithm would return the complete set of relevant documents, and nothing but those. Formally, both precision and recall would equal 1. However, this is generally not the case, and there is some form of trade-off between precision and recall. Preferably, the precision value is quite high for all recall levels. However, most retrieval algorithms introduce a growing number of irrelevant documents into the result set with growing recall, i.e., precision generally diminishes with increasing recall rates. In fact, a recall of 1 can easily be reached by retrieving the complete set of documents, which will naturally include all relevant documents. Furthermore, in a scenario where the number of service offers is very high, relevant services should be mentioned first in the query result set. Hence, it is desirable to achieve a high precision on relatively low recall levels. As the number of relevant services might be one day as large and fast-changing as the number of Web sites is today, it seems likely that a high precision is more important to the service requester than a high recall as completeness of found service offers might be very difficult to achieve and not even desirable [136, 180].

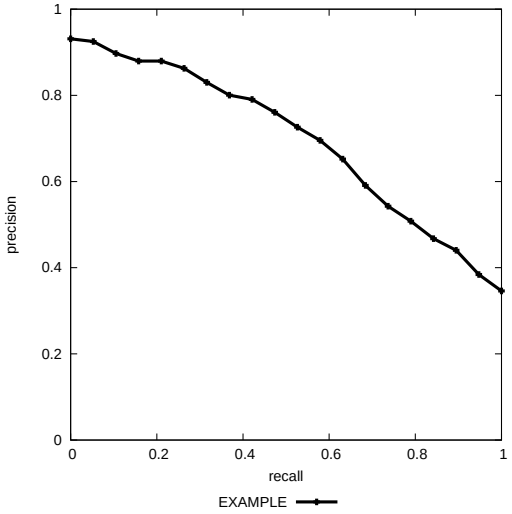


Figure B.1: Recall-Precision Curve (Example)

In order to account for the relationship between and mutual significance of precision and recall, these measures should always be cited and interpreted pairwise. Furthermore, additional metrics which incorporate recall and precision will be used [89, 132, 176]:

- $Precision_i = \frac{1}{|Q|} \times \sum_{q \in Q} \max\{P_O | R_O \geq Recall_i \wedge (R_O, P_O) \in O_q\}$ with O_q representing the set of recall-precision value pairs of the relevant documents for query q, which are determined by a stepwise comparison of the sorted result set in descending order with the corresponding relevance set from SAWSDL-TC [132]. These values are the foundation for so-called recall-precision curves which depict both precision and recall in a two-dimensional diagram. An example is given in Figure B.1 – as it can be seen the precision is quite high for low recall values but declines to the point where all relevant services have been found, i.e., the recall is 1.0. SME2 uses equidistant steps $\frac{n}{\lambda}, n = 1 \ldots \lambda$ for the single recall levels. Here, we made use of the default value $\lambda = 20$ in order to accomplish comparability with evaluation results of other matchmakers. Precision values are *interpolated* for the depiction in recall-precision curves. The interpolated precision at a certain recall level r is defined as the highest precision value found for any recall level $r' \geq r$ [176]. This way, it is, e.g., possible to define a precision value for $r = 0$.

- The Average Precision (AP) is used to provide a single-valued rating of a matchmaker for a single query result set. $AP = \frac{1}{|R|} \times \sum_{r=1}^{|L|} isrel(r) \frac{count(r)}{r}$ where R is the set of relevant documents to the particular query and L the set of found services. $isrel(r) = 1$ if $r \in R$, 0 else. $count(r)$ indicates the number of relevant items so far, i.e., $count(r) = \sum_{i=1}^{r} isrel(i)$ [140]. The *mean* AP is used in order to give one value for the matchmaking results regarding all queries applied. In the S3 Contest, this value is used in order to determine the best matchmaker regarding retrieval performance.

- To measure the precision on relatively low recall levels, Precision at k (P(k)) is employed. This value specifies the precision for the first ranked k data items; common values are, e.g., $k = 5$ or $k = 10$ (named "P(5)" or "P(10)") [176, 254].

- R-Precision (RP) defines the precision for a certain number of returned (ranked) results to a specific query. Where P(k) defines one k for all queries, in RP this k differs from query to query and is equivalent to the relevant services for this query (i.e., the size of the result set provided in SAWSDL-TC). For RP, recall and precision are the same, hence it is equivalent to the *break-even point* [176].

AP, RP, and P(k) are *macro*-averaged over all queries. I.e., the results for every query are averaged giving an equal weight to each query. In contrast, it would also be possible to make use of *micro*-averaged values where an equal weight is given to each service from a result set. This way, queries would be weighted

unequally [176]. However, in our opinion the performance regarding each query is equivalent and should be weighted likewise.

Together, recall, precision, P(5) and P(10), recall-precision curves, AP, and RP allow to compare the retrieval correctness as well as completeness with different scenarios in mind. While, e.g., P(5) is suitable to measure the retrieval performance if regarding automatic service invocation which needs very good precision results on low recall levels, recall-precision curves and precision values can be used in order to observe the completeness of the result set. While high precision at low recall levels is primarily needed in scenarios where service discovery is done just prior to the actual service invocation, in a second scenario where services might be manually adapted and integrated into, e.g., a service workflow, services which are not regarded in P(5) or P(10) might also be considered.

B.4 Cross-validation

Generally, if computing predictions of optimal values, as it is done by the application of OLS in this thesis, the training set (i.e., the data set the prediction is based on) needs to be clearly separated from the test data (i.e., the data set used in the evaluation) in order to get valid evaluation results. Here, *cross-validation* is a well-accepted method [187, 225, 256].

k-fold cross-validation determines the classification of a data set into k partitions or subsamples. k-1 of the subsamples are used for training, while the remaining subsample is used for testing. This process is repeated k times and also known as *Leave-one-out* cross-validation. This way, it is possible to see how the results of the prediction (here: OLS estimator) can be applied to an independent data set which has not been used in the training phase. In the work at hand, the subsamples are given by the queries and relevance sets from SAWSDL-TC (cp. Appendix B.2.2). Each query is once used for testing and 25 times (as $k = 26$) for the training of the OLS estimator. In each run, the matchmaking results for one query are computed.

However, we have to slightly depart from the assumption of independent data sets (subsamples), as some service offers are allocated to more than one relevance set. This is done to meet the fact that some of the service requests' respective relevance sets are not mutually disjunct. As a concession to this, all service offers that are relevant to at least one service request besides the one being currently validated are taken into consideration in the training stage. Furthermore, not all service offers are part of at least one result set. Hence, not every service offer is regarded in the training phases.

B.5 Evaluation of Runtime Performance

The runtime performance of the matchmakers presented in Chapter 3 heavily depends on which parts of the matchmaking process can be conducted during publication time and which parts need to be done during the actual runtime.

We make use of a scenario that is applied, e.g., by Kourtesis and Paraskakis [145]: Services need to be registered at a service repository. During publication time, an ATSM model is created for each service upon registration. Further, each new service offer is matched against each previously registered service. As a result, all pairs of service offers (and their respective components) have been mutually matched upon the completion of the registration phase. Both COM4SWS.KOM and LOG4SWS.KOM were configured to conduct matching on all levels; i.e., level weights were set to a value greater than 0. Lastly, the fallback strategy was enabled during the training phase. This guarantees that all service offers are available as ATSM models and the caches (cp. Section 3.2.2) are populated to a maximum level, based on all available information in the registration phase. Lastly, the matchmakers have been configured to create the input data for the OLS in the training phase.

The actual runtime evaluation then operates on the existing ATSM models, the populated caches, and the previously generated OLS data. In case of the SAWSDL-TC test collection, there exists a corresponding service offer for each of the 26 queries. Thus, at runtime, no misses occurred in any of the caches; i.e., all data required for matching was readily available in the caches. However, the queries had to be transformed into an ATSM model. In fact, this is the most time-intensive part of query processing, accounting for approximately 80% of the overall query response time. The remaining part can be attributed to the

actual matching and the determination of weights and numerical equivalents through the application of OLS.

For each individual variant of LOG4SWS.KOM and COV4SWS.KOM as presented in Section 3.4, a total of six consecutive runs were conducted, and the results were aggregated using the arithmetic mean. Runs were executed on a Intel Core 2 Duo T7300 (2Ghz) notebook with 3 Gbyte of RAM, operating with Microsoft Windows XP. All background tasks (such as virus scanners or indexing programs) and non-required system services were disabled.

However, there were still some fluctuations in runtime performance measurements that can be attributed to operation system background activities. To compensate potential outliers, we calculate three different average values:

Median and Median Absolute Deviation (MAD) for all six test runs.

Arithmetic mean and standard deviation for all six test runs.

Arithmetic mean and standard deviation for a trimmed evaluation result set where the best and worst results (in terms of runtime performance) were discarded.

The AQRT as presented in Section 3.4 refers to the median of all six evaluation runs per version/variant of LOG4SWS.KOM and COV4SWS.KOM. Further query response time results are presented in Appendix C.4.

C Further Evaluation Results

This section enhances the evaluation results of the matchmaking approaches presented in Chapter 3 by further numbers. While in the evaluation in Section 3.4 macro-averaged AP and RP values have been presented, the following diagrams present the evaluation results for each single query from the test data set (Sections C.1 and C.2). Afterwards, the results from the Friedman tests conducted for the different versions of LOG4SWS.KOM and COV4SWS.KOM are presented in Section C.3. Furthermore, the runtime performance numbers presented in Section 3.4 are enhanced by further values. Finally, Section C.5 presents the precision values for the distinct recall levels presented in Figures 3.7–3.9.

C.1 Comparison of Average Precision per Query

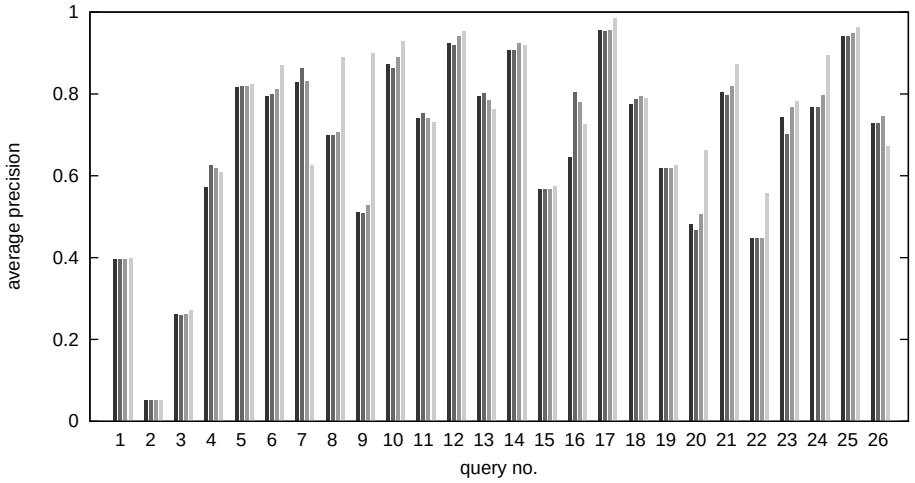
Analog to the illustration in Figures 3.7, 3.8, and 3.9, Figures C.1 to C.4 show the AP for each query as evaluated with SAWSDL-TC (cp. Section 3.4).:

- Figures C.1a to C.1c show the AP values per query arranged for the three versions of LOG4SWS.KOM.
- Figures C.2a to C.2d show the AP values per query arranged for the four variants of LOG4SWS.KOM.
- Figures C.3a to C.3d show the AP values per query arranged for the four versions of COV4SWS.KOM.
- Figures C.4a to C.4f show the AP values per query arranged for the six variants of COV4SWS.KOM.

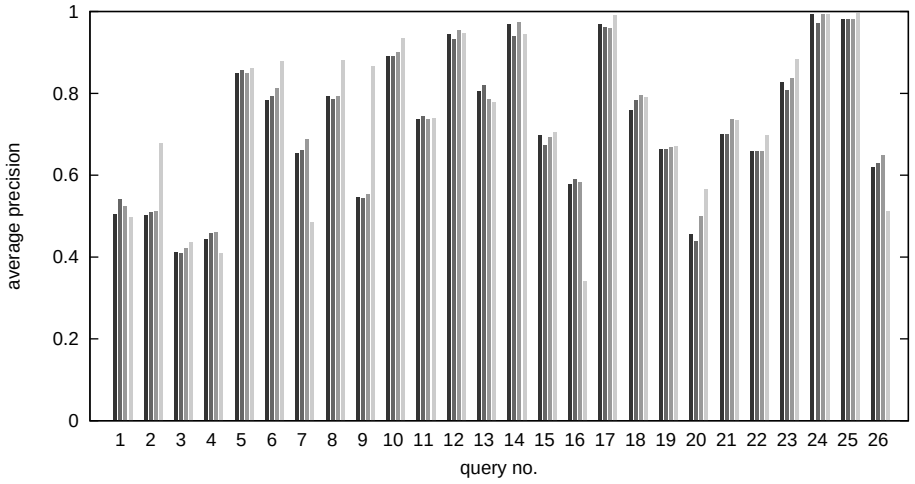
As it can be seen from Figures C.1a to C.1c, the average precision for single queries is rather heterogeneous. Regarding Version 1, Variant 1d improves the AP for a predominant share of queries (16 out of 26). However, most improvements are rather small and results do not vary too much – for 14 out of 26 queries, similar AP values can be determined. Similar observations can be made for Versions 2 and 3, where Variant 2d improves 13 out of 26 results and Variant 3d 12 out of 26, but for the better part, the AP values are quite similar. There are always some queries where the OLS-based determination of numerical equivalents leads to the worst results or worse results than Variant C. This is especially the case for Variant 3d and explains why for Version 3, Variant D does not provide the best results and the result differences are rather small. Which queries do not gain from the OLS-based weightings is varying to some extent. This is not surprising and supports the assumption that there is no generally best matchmaking method for Web services. Furthermore, it can be seen that for many queries, the incorporation of manually adapted numerical equivalents of DoMs (i.e., Variants B and C), improves matchmaking results in comparison to the numerical equivalents proposed by Syeda-Mahmood et al. (i.e., Variant A) [244].

Regarding the different variants, Figures C.2a to C.2d show that the data is even more heterogeneous. Regarding Variant A, almost half of the queries (11 out of 26) show similar AP values for all three versions, but it should be noted that here, only three datasets are compared, while for the different versions, four datasets need to feature similar values. For the other queries, the three versions perform unequally. Similar (but not equal) patterns can be observed for Variants B, C, and D.

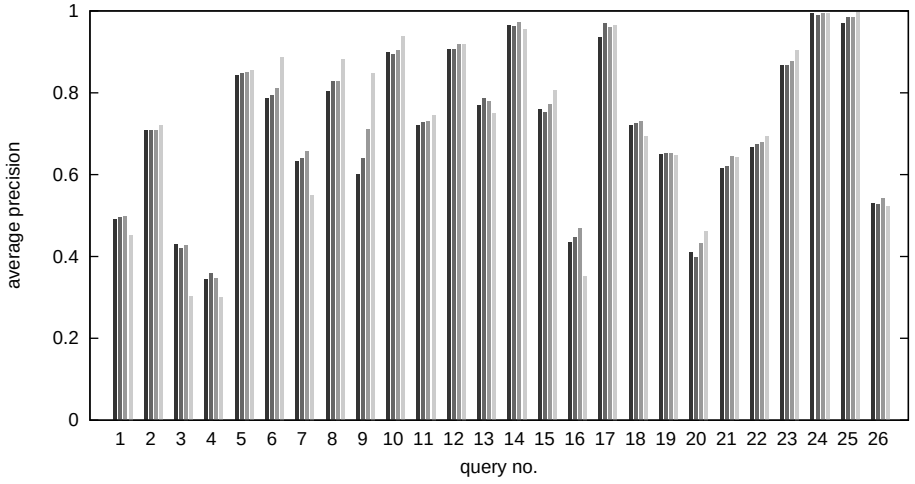
Regarding COV4SWS.KOM, it is even more difficult to determine a clear pattern – let alone because of the higher number of variants. However, there are some results worth noting: For Version 1, 14 out of 26 queries feature similar results for Variants E and F, for Version 2, 13 out 26, for Version 3, 22 out of 26, and for Version 4, 17 out of 26. Hence, we can deduct, that sim_{PL} and sim_{MC} provide very similar results for a large part of queries – this is not surprising, as sim_{MC} heavily depends on the path length. As a second result, it can be determined that the MC-based AP values of sim_{Resnik} and sim_{Lin} differ surprisingly often, if taking the quite similar results from Table 3.5 into account; nevertheless, in most cases, the results are better for the corpus-based variants (A and C). Thus, we can deduct that the corpus-based probability values needed for these similarity measures are sufficient for most queries, but in some cases, the MC-based probability values are better suited.



(a) Version 1

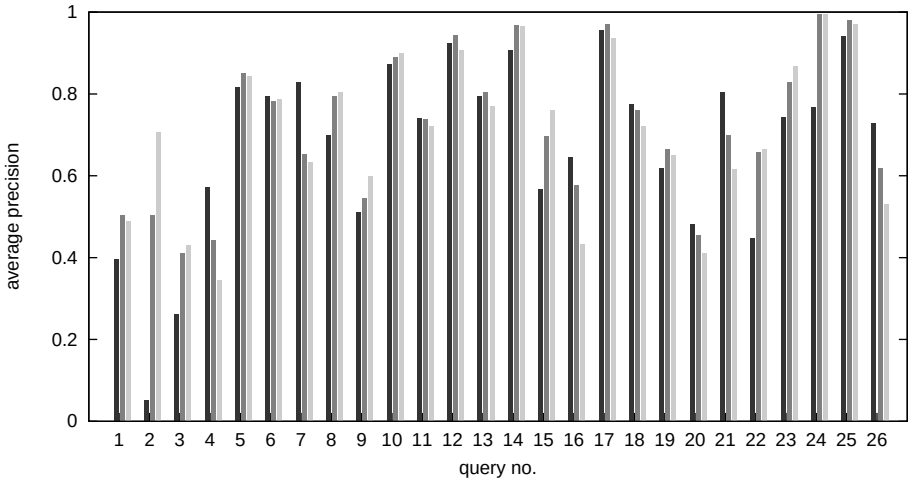


(b) Version 2

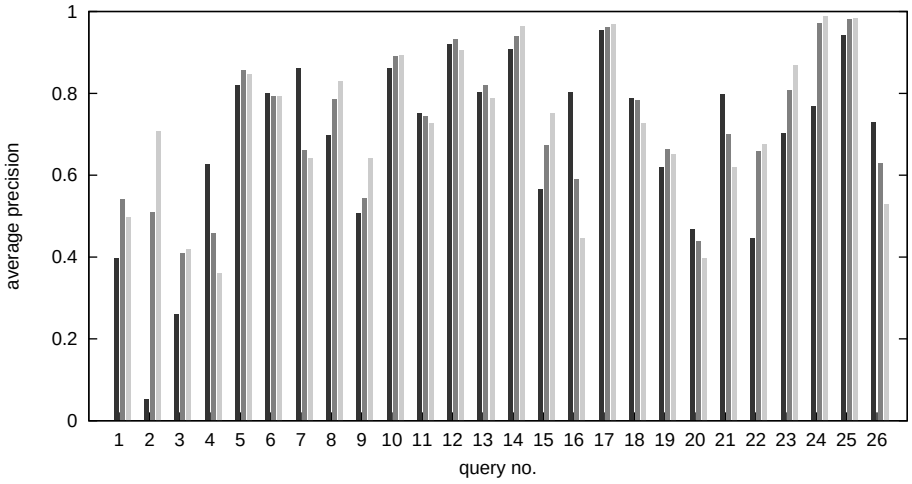


(c) Version 3

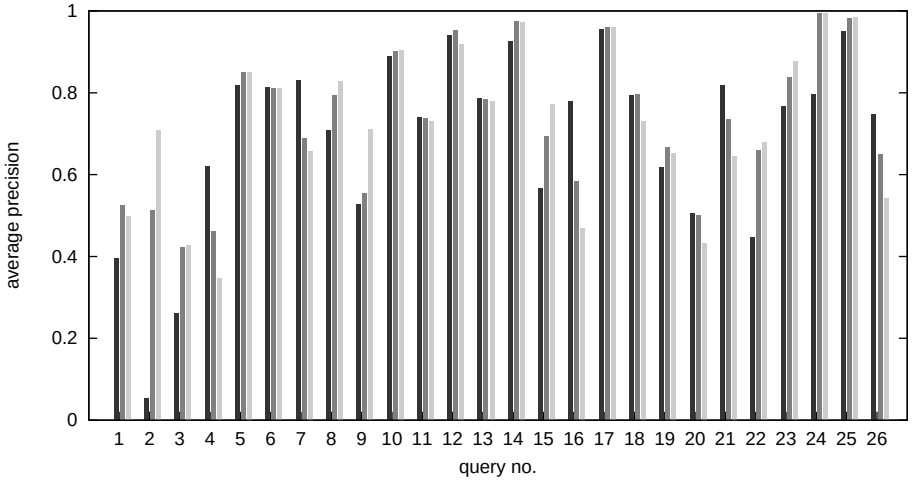
Figure C.1: Average Precision for LOG4SWS.KOM per Query – Versions



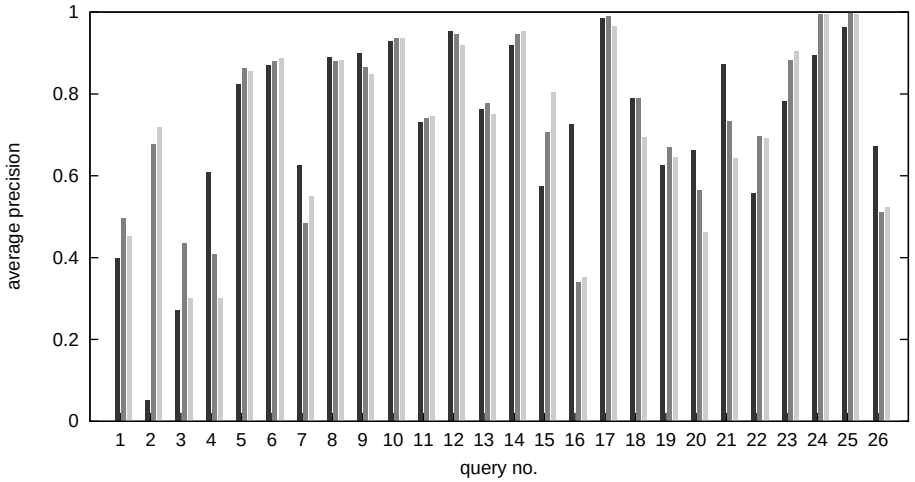
(a) Variant A



(b) Variant B

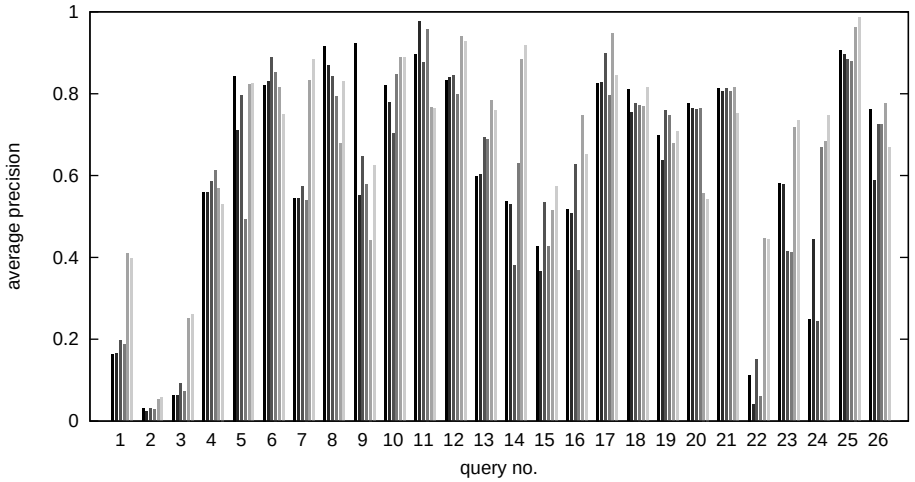


(c) Variant C

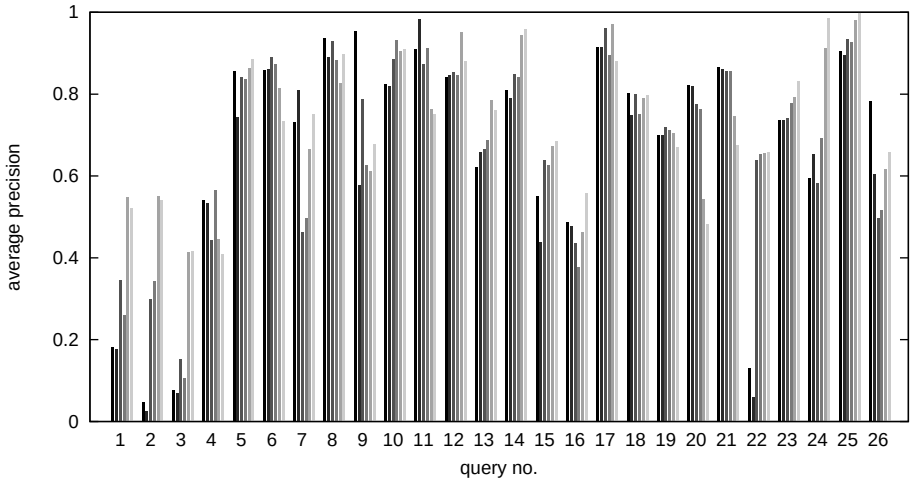


(d) Variant D

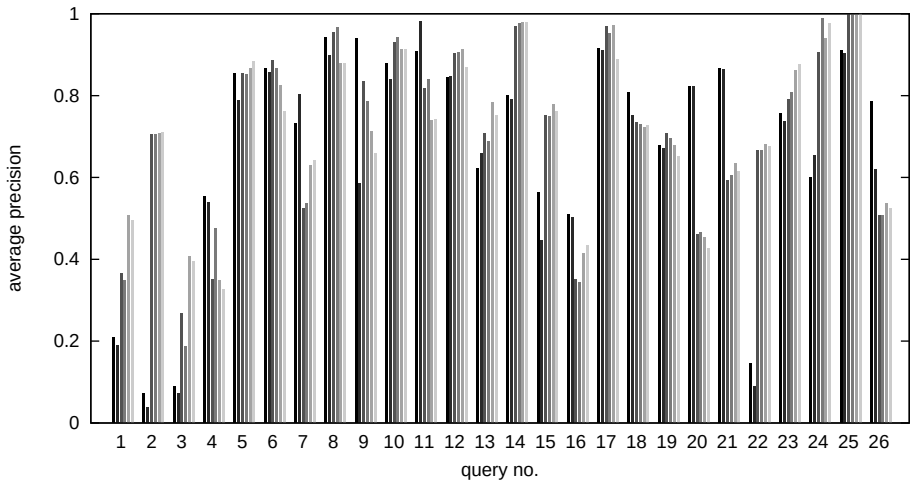
Figure C.2: Average Precision for LOG4SWS.KOM per Query – Variants



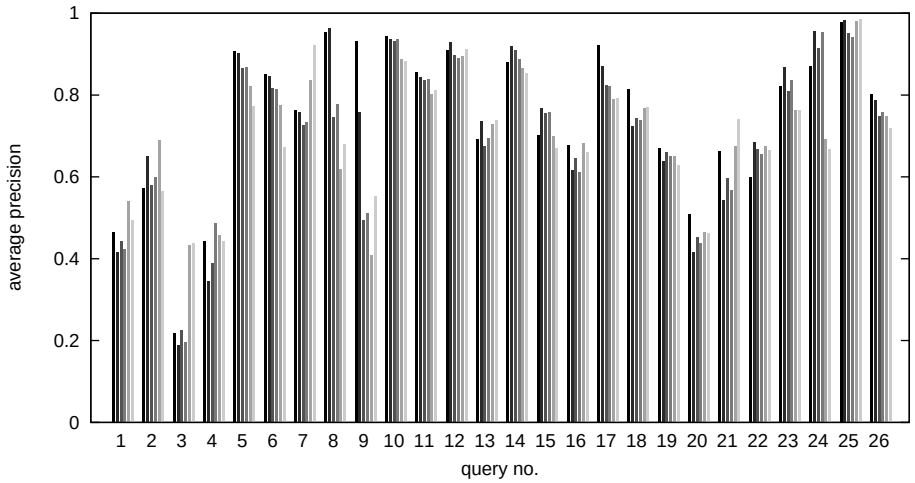
(a) Version 1



(b) Version 2



(c) Version 3



(d) Version 4

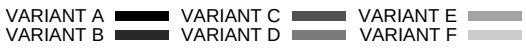
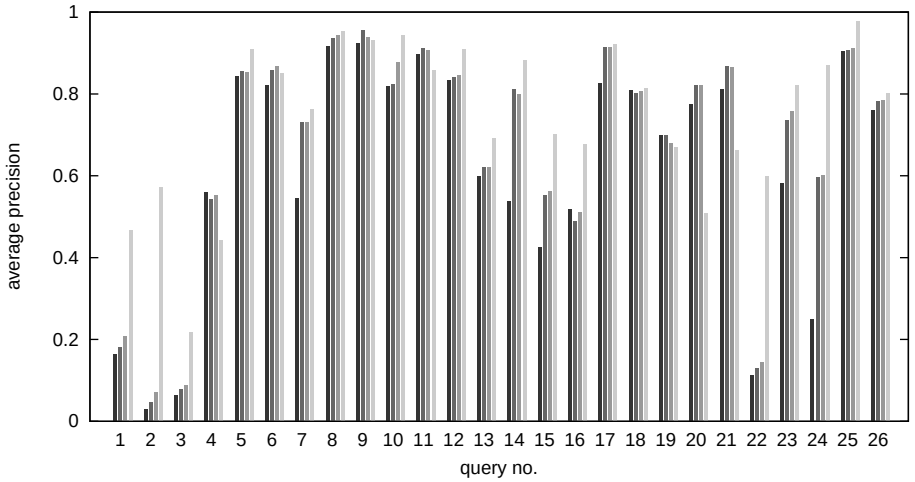
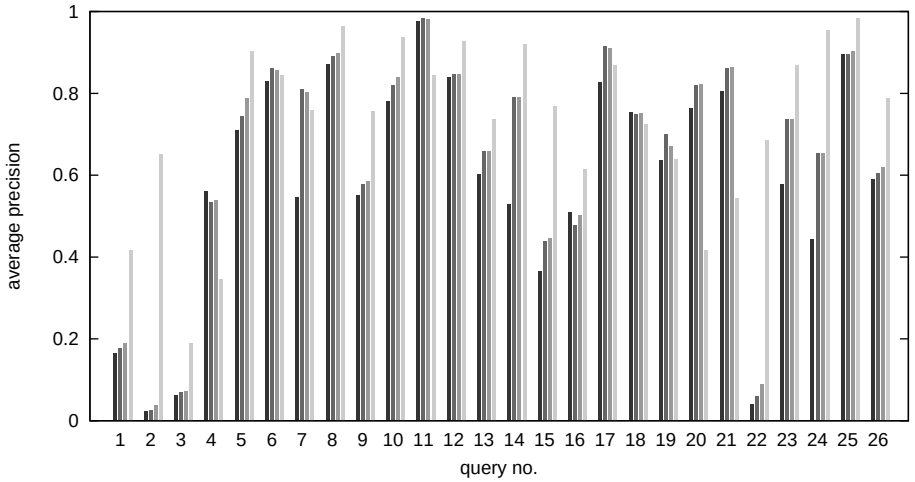


Figure C.3: Average Precision for COV4SWS.KOM per Query – Versions

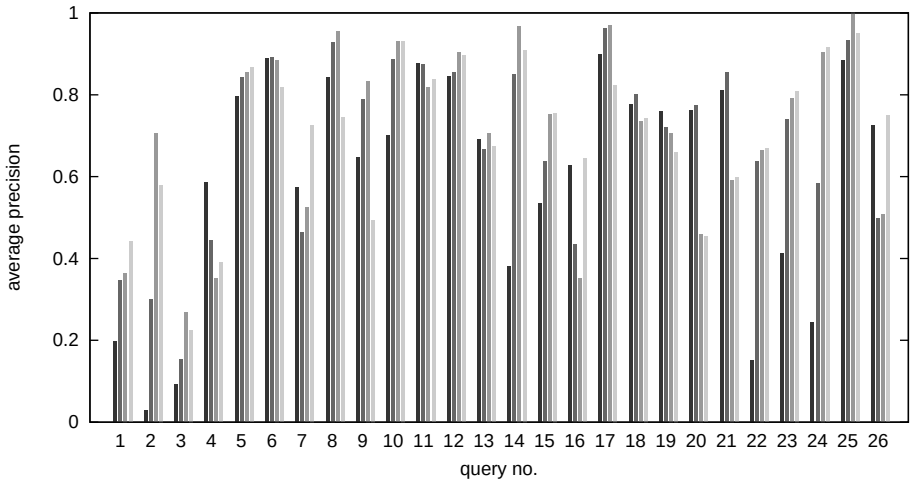
Regarding the different variants of COV4SWS.KOM, the results from Section 3.4.2 are approved: For Variant A, Version 4 clearly outperforms all other versions with the best values for 19 out of 26 queries. Version 1 performs a little bit worse than Versions 2 and 3, which possess similar values. A similar pattern can be recognized for Variant B (17 out 26). In particular, there are a number of queries for both variants where the integration of OLS-based weightings leads to a massive advancement in evaluation results. In contrast, Variants C and D feature much more heterogeneous AP values: Again, Version 1 clearly performs worst, but for the remaining versions, there is no clear pattern – resulting in the relatively similar AP values presented in Table 3.5. Apart from very few queries (8, 9, 12, to some extent 24 and 26), Variants E and F feature similar value patterns in Figures C.4e and C.4f. So, the similar matchmaking performance of these variants can be observed on query-level, too.



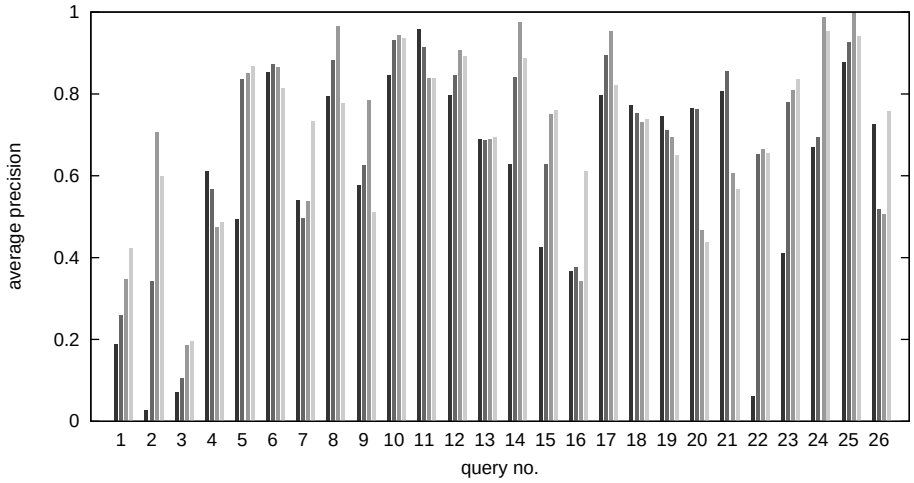
(a) Variant A



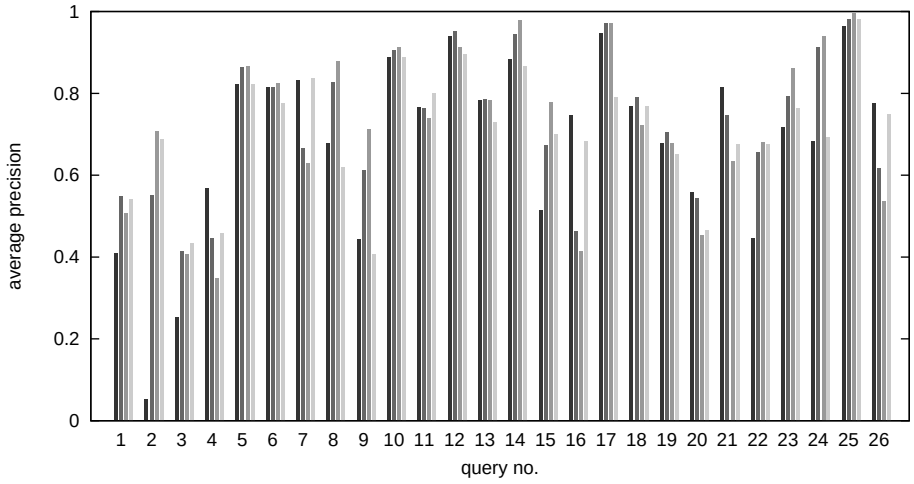
(b) Variant B



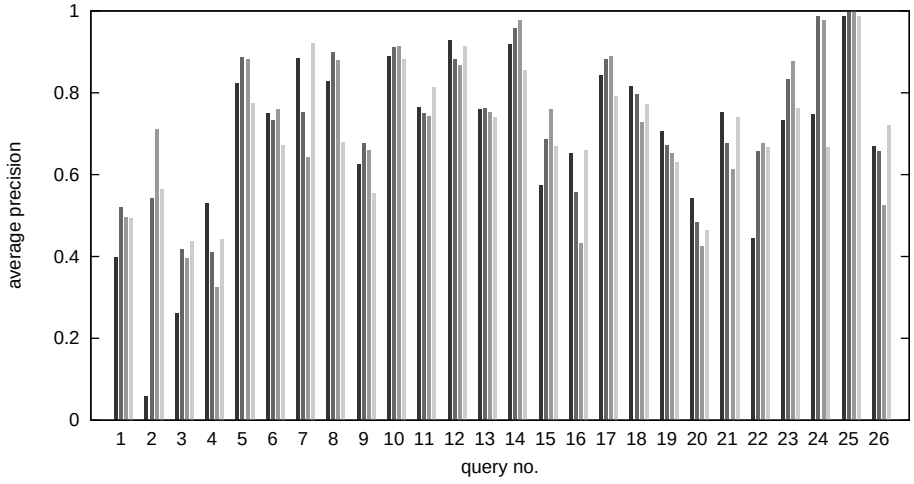
(c) Variant C



(d) Variant D



(e) Variant E



(f) Variant F

Figure C.4: Average Precision for COV4SWS.KOM per Query – Variants

C.2 Comparison of R-Precision Values per Query

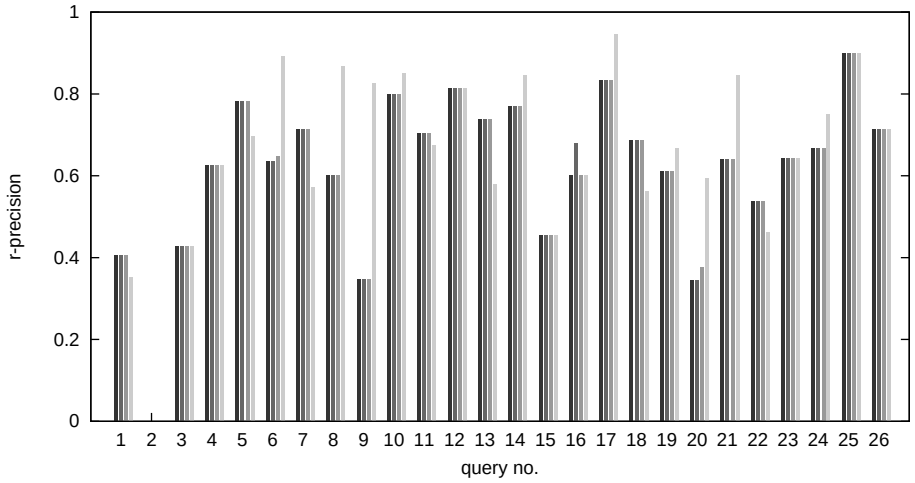
Figures C.5 to C.8 show the R-Precision for the different versions/variants of LOG4SWS.KOM and COV4SWS.KOM:

- Figures C.5a to C.5c show the RP values per query arranged for the three versions of LOG4SWS.KOM.
- Figures C.6a to C.6d show the RP values per query arranged for the four variants of LOG4SWS.KOM.
- Figures C.7a to C.7d show the RP values per query arranged for the four versions of COV4SWS.KOM.
- Figures C.8a to C.8f show the RP values per query arranged for the six variants of COV4SWS.KOM.

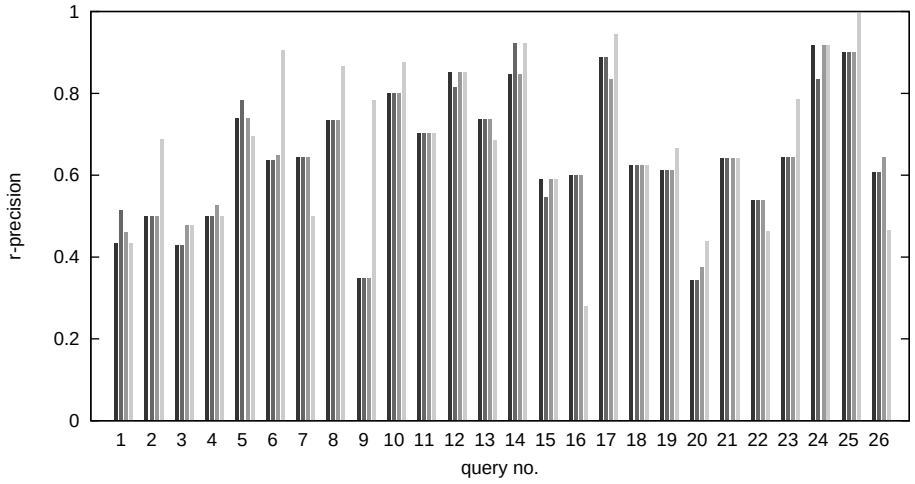
Naturally, the number of potential values for every single query is smaller than those observed for the AP values in the last section.

Interestingly, for Version 1 of LOG4SWS.KOM, except for Query 16, every query features the same RP value for Variants A and B; for Variant A, B, C, the values are the same except for Queries 6, 16, and 20. Hence, we can assume that the different "manual" weightings have a rather small impact regarding Version 1 if regarding the RP. The differences are backed up by the similar progressions of the recall-precision curves for Version 1 depicted in Figure 3.8 up to a recall level of about 0.6. This observation is supported by the RP, P(5), and P(10) values from Table 3.6. The same applies to Versions 2 and 3 to a smaller extent (Version 2: 14 queries share the same values for the first three variants, six further queries share the same values for the first two variants; Version 3: 17 queries share the same values for the first three variants, four further queries share the same values for the first two variants). Over all versions, the OLS-based Variant D performs best for most queries, certifying the results regarding RP, P(5), and P(10) from Table 3.6. When comparing the different variants of LOG4SWS.KOM (cp. Figure C.6), the differences between the single versions are much easier to observe, as the values for single queries differ more substantially. Hence, we can assume that the differences in matchmaking results can be primarily attributed to the different variants, i.e., numerical equivalents of DoMs, than the different weights put into effect for the service abstraction levels (at least regarding the R-Precision).

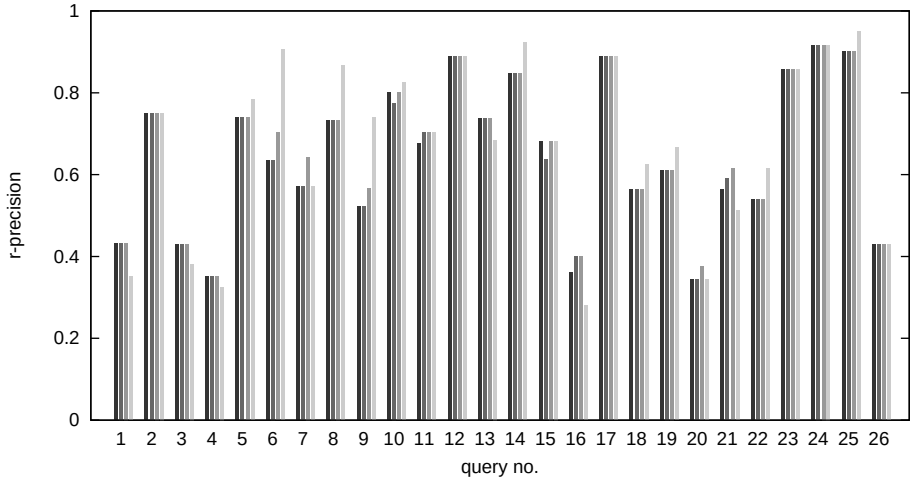
Again, the values for COV4SWS.KOM are more heterogeneous. Nevertheless, there are some outcomes worth mentioning: First of all, for Queries 1–3 and 22, the integration of all service abstraction levels massively increases the RP, which is very low for Variants 1a–1d. Second, the heterogeneity already mentioned in Section 3.4.2 can be observed again: While for Variants A and B, 8 respectively 12 out of 26 queries possess the same RP for the first three versions, this is only the case for 3 respectively 1 out of 26 queries for Variants C and D. Here, the disproportionately high influence of the non-normalized Variants A and B can be observed, leading to similar values for the first three versions and being resolved by the OLS-based weighting in Variant 4a. For Variant E, only two queries share the same RP for Versions 1–3, while this is the case for six queries in Variant F. Interestingly, the RP values of Variants E and F vary much more than the AP values – while the overall picture in Figures C.4e and C.4f is very similar, it is not possible to identify a common pattern in Figures C.8e and C.8f. This can be traced back to the fact that the number of potential RP values is much smaller and variances attract attention much easier. Hence, the AP seems to be a more appropriate performance metric to assess general similarities between two variants.



(a) Version 1

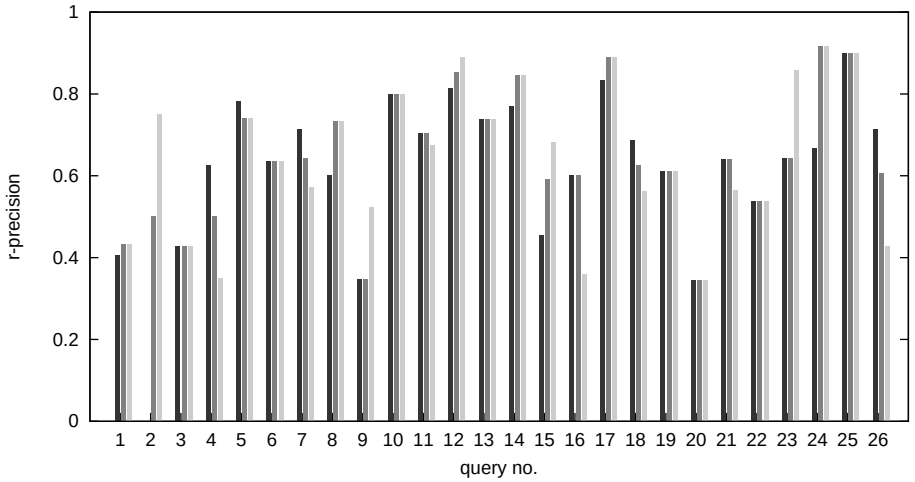


(b) Version 2

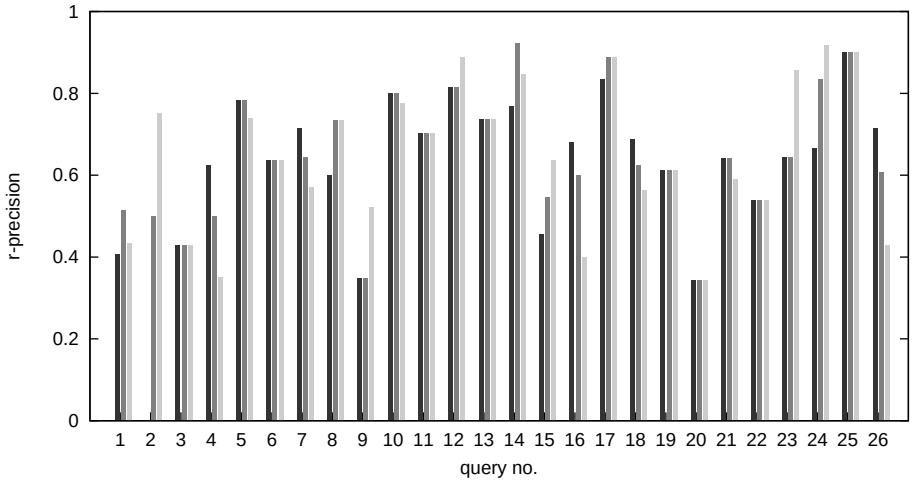


(c) Version 3

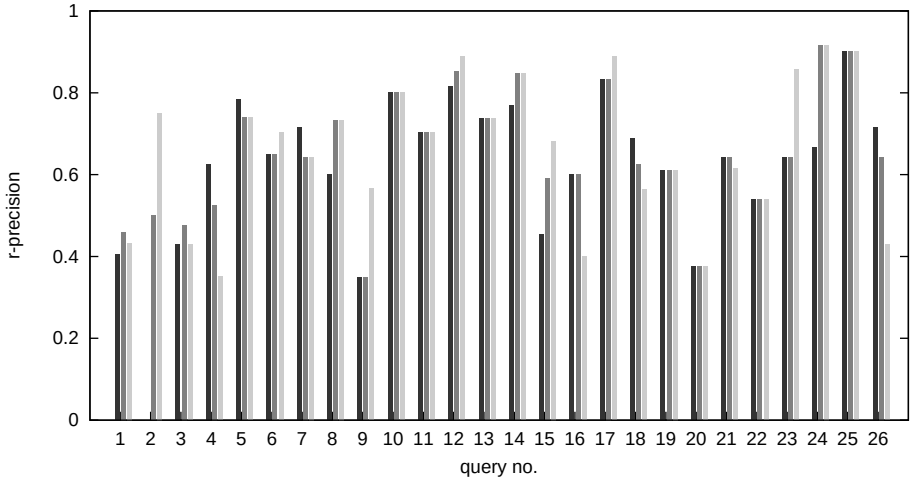
Figure C.5: R-Precision for LOG4SWS.KOM per Query – Versions



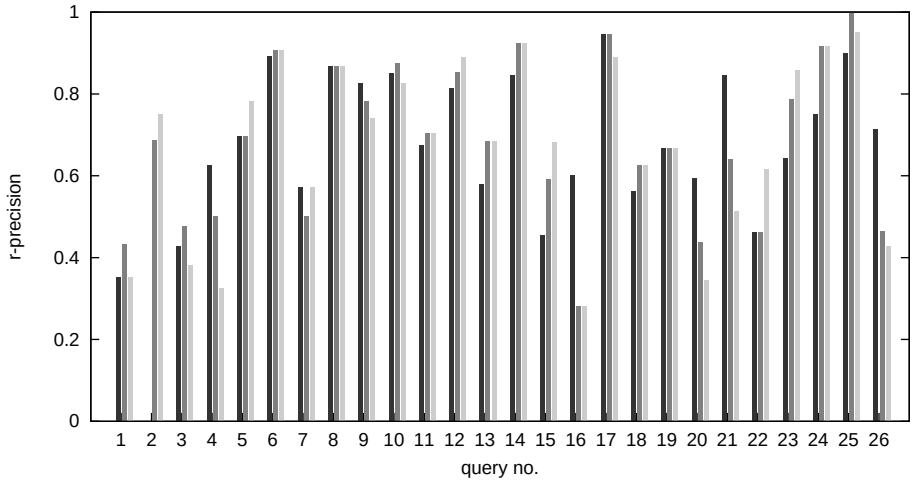
(a) Variant A



(b) Variant B



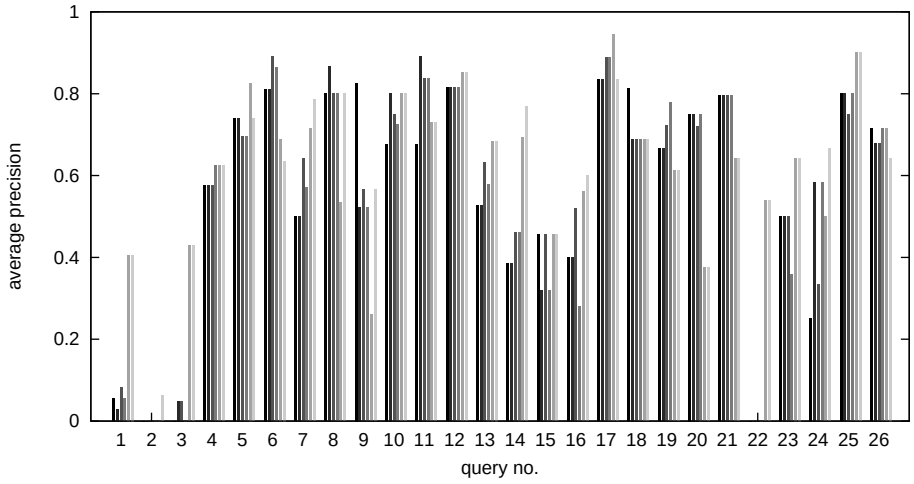
(c) Variant C



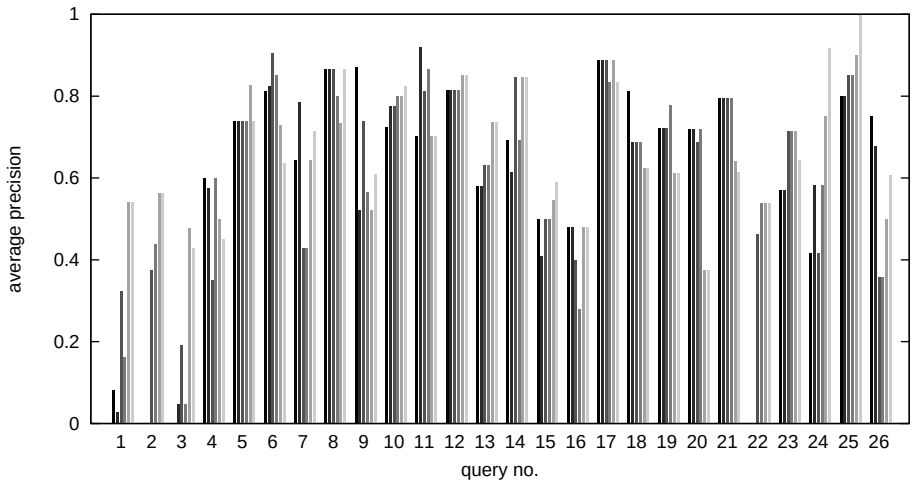
(d) Variant D

VERSION 1 VERSION 2 VERSION 3

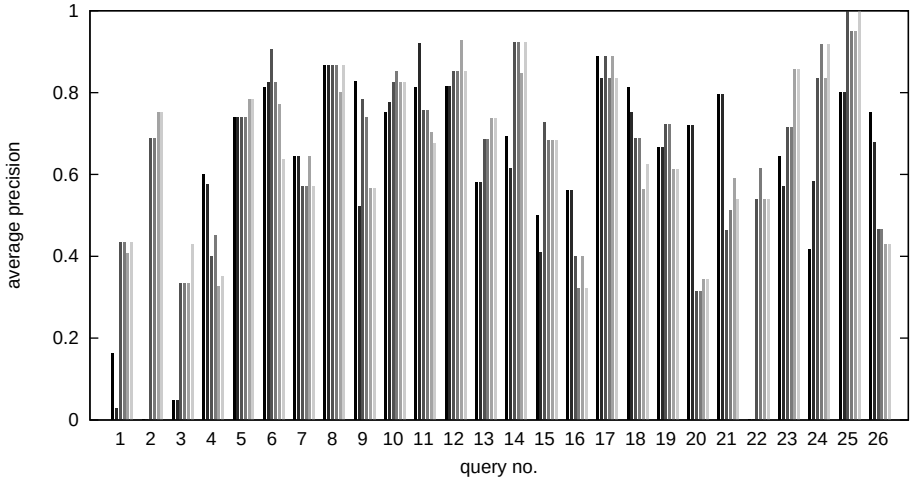
Figure C.6: R-Precision for LOG4SWS.KOM per Query – Variants



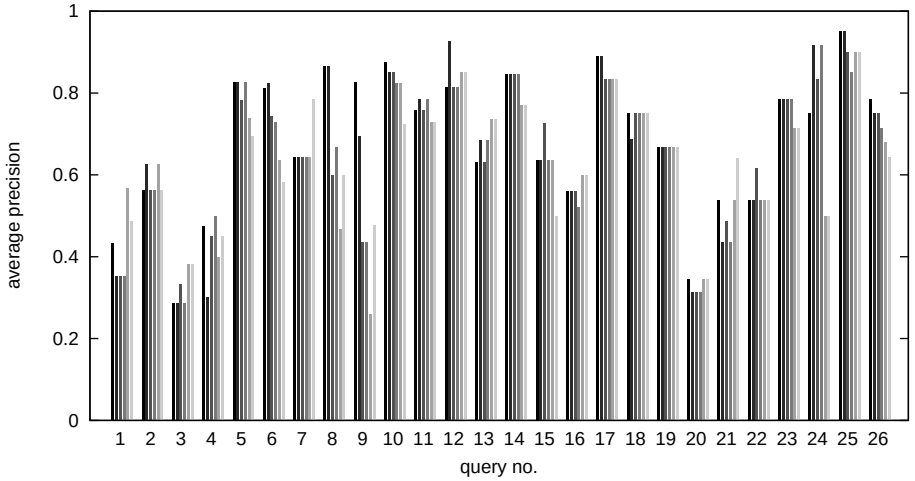
(a) Version 1



(b) Version 2

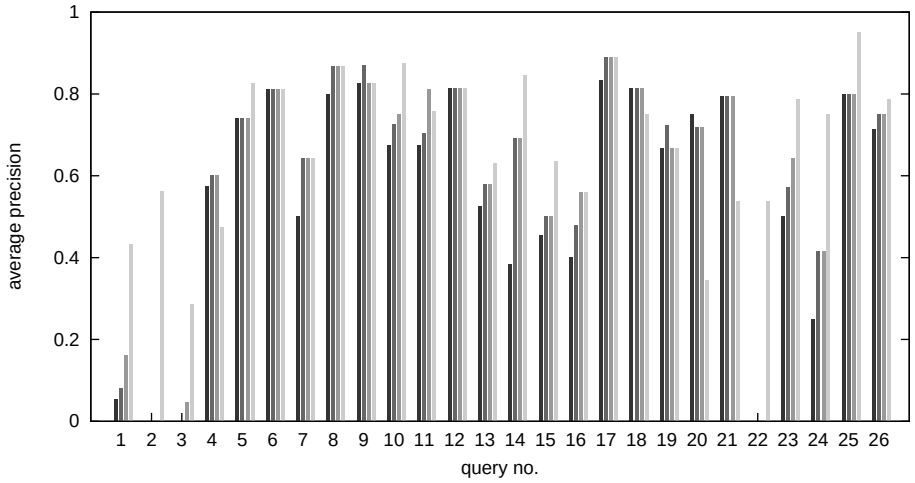


(c) Version 3

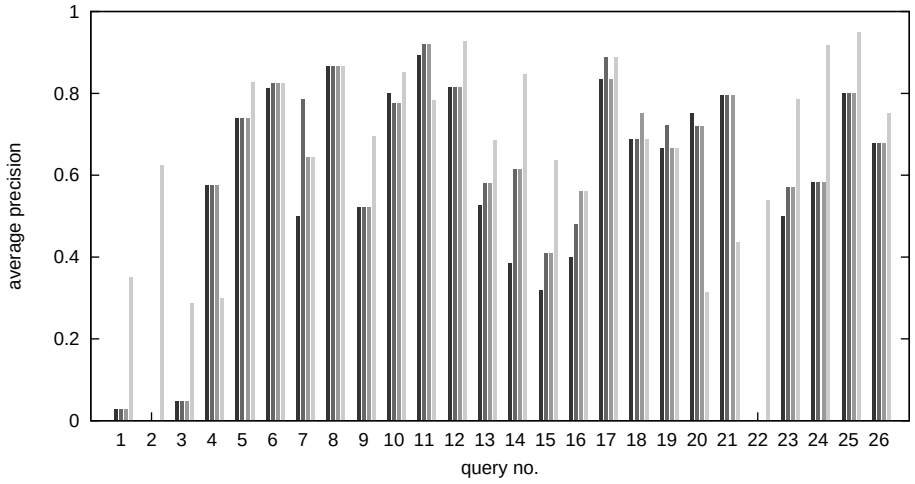


(d) Version 4

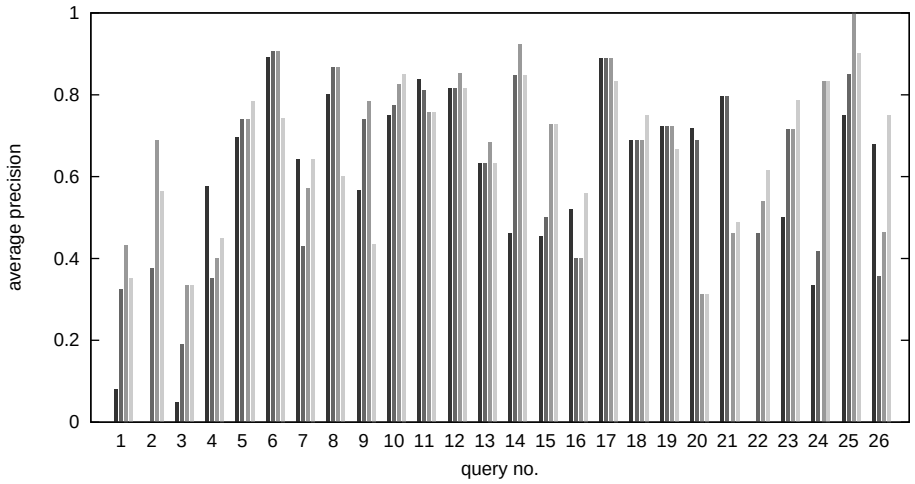
Figure C.7: R-Precision for COV4SWS.KOM per Query – Versions



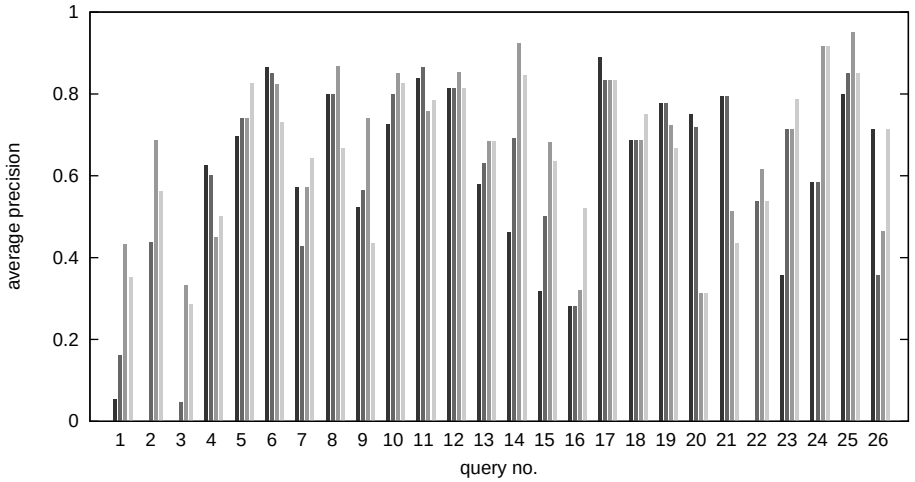
(a) Variant A



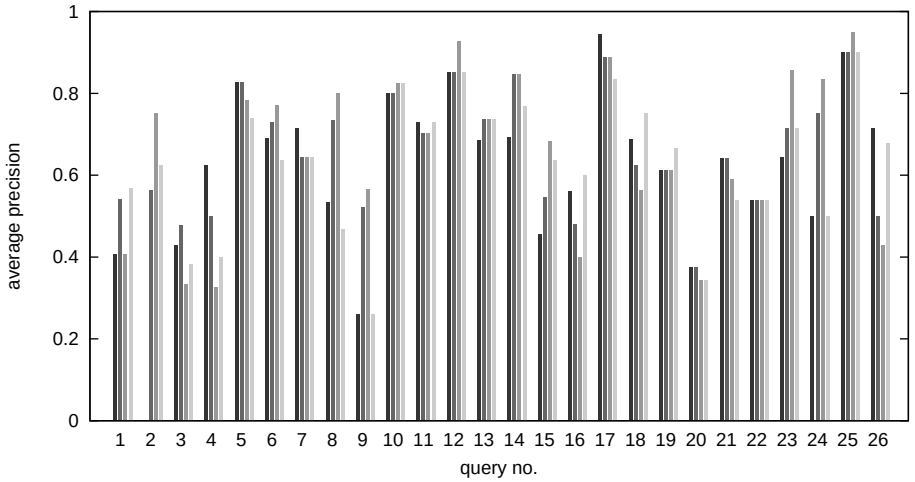
(b) Variant B



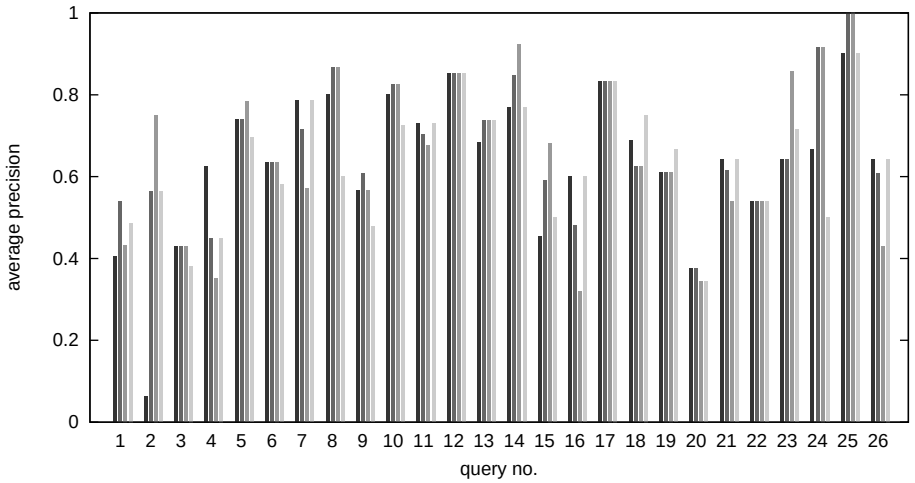
(c) Variant C



(d) Variant D



(e) Variant E



(f) Variant F

Figure C.8: R-Precision for COV4SWS.KOM per Query – Variants

C.3 Friedman Tests

Tables C.1 to C.7 show the results for the Friedman tests conducted for the different evaluation results (AP per query) regarding the different versions/variants of LOG4SWS.KOM respectively COV4SWS.KOM. The *Overall* value is the test result if all variants of a version are taken into account.

Table C.1: Friedman Test for LOG4SWS.KOM, Version 1

$p =$	Variant 1a	Variant 1b	Variant 1c	Variant 1d
Variant 1a	X	0.824	0.000	0.000
Variant 1b	X	X	0.011	0.004
Variant 1c	X	X	X	0.118
Variant 1d	X	X	X	X
Overall: 0.000				

Table C.2: Friedman Test for LOG4SWS.KOM, Version 2

$p =$	Variant 2a	Variant 2b	Variant 2c	Variant 2d
Variant 2a	X	0.846	0.000	0.025
Variant 2b	X	X	0.007	0.015
Variant 2c	X	X	X	0.327
Variant 2d	X	X	X	X
Overall: 0.000				

Table C.3: Friedman Test for LOG4SWS.KOM, Version 3

$p =$	Variant 3a	Variant 3b	Variant 3c	Variant 3d
Variant 3a	X	0.071	0.000	0.327
Variant 3b	X	X	0.000	0.444
Variant 3c	X	X	X	0.846
Variant 3d	X	X	X	X
Overall: 0.000				

Table C.4: Friedman Test for COV4SWS.KOM, Version 1

$p =$	Variant 1a	Variant 1b	Variant 1c	Variant 1d	Variant 1e	Variant 1f
Variant 1a	X	0.071	0.703	0.444	0.048	0.118
Variant 1b	X	X	0.048	0.228	0.000	0.004
Variant 1c	X	X	X	0.015	0.118	0.247
Variant 1d	X	X	X	X	0.048	0.015
Variant 1e	X	X	X	X	X	1.000
Variant 1f	X	X	X	X	X	X
Overall: 0.000						

Table C.5: Friedman Test for COV4SWS.KOM, Version 2

$p =$	Variant 2a	Variant 2b	Variant 2c	Variant 2d	Variant 2e	Variant 2f
Variant 2a	X	0.025	0.703	0.247	0.444	0.444
Variant 2b	X	X	0.048	0.118	0.048	0.048
Variant 2c	X	X	X	0.444	0.048	0.247
Variant 2d	X	X	X	X	0.118	0.048
Variant 2e	X	X	X	X	X	0.444
Variant 2f	X	X	X	X	X	X
Overall: 0.029						

Table C.6: Friedman Test for COV4SWS.KOM, Version 3

$p =$	Variant 3a	Variant 3b	Variant 3c	Variant 3d	Variant 3e	Variant 3f
Variant 3a	X	0.004	0.118	0.444	0.703	1.000
Variant 3b	X	X	0.048	0.048	0.247	0.703
Variant 3c	X	X	X	0.846	0.247	0.559
Variant 3d	X	X	X	X	0.444	0.703
Variant 3e	X	X	X	X	X	0.247
Variant 3f	X	X	X	X	X	X
Overall: 0.232						

Table C.7: Friedman Test for COV4SWS.KOM, Version 4

$p =$	Variant 4a	Variant 4b	Variant 4c	Variant 4d	Variant 4e	Variant 4f
Variant 4a	X	0.247	0.004	0.048	0.247	0.048
Variant 4b	X	X	0.048	0.015	0.247	0.247
Variant 4c	X	X	X	1.000	0.703	0.703
Variant 4d	X	X	X	X	1.000	0.703
Variant 4e	X	X	X	X	X	0.444
Variant 4f	X	X	X	X	X	X
Overall: 0.031						

C.4 Comparison of Average Response Time per Query

Table C.8: Comparison of Average Query Response Times for LOG4SWS.KOM (in ms)

#	Median	MAD	Arithmetic Mean(6)	Standard Deviation(6)	Arithmetic Mean(4)	Standard Deviation(4)
1a	2137.37	168.04	2111.23	229.98	2135.81	173.76
1b	2187.69	29.79	2105.96	153.08	2135.82	125.29
1c	2235.48	119.77	2102.57	306.00	2150.02	257.79
1d	2217.52	185.56	2174.46	286.81	2217.42	203.88
2a	2316.90	32.90	2293.24	68.11	2304.06	40.28
2b	2287.62	76.62	2276.89	106.26	2287.81	87.43
2c	2405.13	67.67	2294.76	234.78	2338.43	157.00
2d	2376.08	1.77	2394.01	43.76	2376.71	2.47
3a	2253.06	135.75	2245.44	159.25	2248.77	151.38
3b	2332.83	98.38	2245.66	258.17	2317.99	94.17
3c	2280.19	69.17	2282.69	67.95	2280.64	61.28
3d	2462.88	16.25	2388.92	141.61	2431.20	72.91

In this section, the runtime performance measure presented in Section 3.4 (i.e., the median) is amended by the MAD, the arithmetic mean, and the standard deviation. Each version/variant of both LOG4SWS.KOM and COV4SWS.KOM has been tested six times regarding runtime performance. To compensate potential outliers, we also calculated the arithmetic mean and the standard deviation for an evaluation test data set where the best and worst query response time for each query have been discarded. These values are named *Arithmetic Mean(4)* respectively *Standard Deviation(4)* as four test runs are regarded. Accordingly, *Arithmetic Mean(6)* and *Standard Deviation(6)* incorporate all (six) test runs. Analog to Tables 3.4 and 3.5, Tables C.8 and C.9 show these values for LOG4SWS.KOM and COV4SWS.KOM (in ms).

Table C.9: Comparison of Average Query Response Times for COV4SWS.KOM (in ms)

#	Median	MAD	Arithmetic Mean(6)	Standard Deviation(6)	Arithmetic Mean(4)	Standard Deviation(4)
1a	2322.19	102.38	2330.54	142.41	2316.88	94.47
1b	2303.15	80.06	2297.66	112.99	2308.22	67.58
1c	2366.33	46.87	2404.92	122.23	2368.13	45.73
1d	2389.38	18.21	2416.20	71.58	2392.90	19.74
1e	2211.67	18.21	2161.47	121.29	2177.25	95.56
1f	2099.77	50.87	2086.04	233.64	2141.92	97.41
2a	2496.46	24.27	2610.84	208.10	2555.63	133.35
2b	2642.29	286.23	2621.68	328.79	2626.68	327.84
2c	2562.00	257.46	2606.71	333.17	2588.35	327.23
2d	2729.56	155.92	2687.58	209.84	2711.20	169.26
2e	2445.50	26.79	2433.23	46.26	2442.38	29.10
2f	2376.75	62.33	2367.58	125.77	2387.92	53.46
3a	2592.50	116.85	2612.78	152.95	2610.21	98.12
3b	2528.52	115.08	2596.88	208.95	2585.90	185.70
3c	2714.12	5.46	2713.23	14.24	2715.77	4.85
3d	2696.08	66.33	2681.28	81.14	2682.13	67.17
3e	2325.12	47.35	2270.39	247.26	2308.38	43.41
3f	2352.90	81.60	2329.20	107.63	2330.38	91.62
4a	2726.54	341.79	2747.26	416.53	2724.72	391.56
4b	2662.79	292.54	2649.11	328.84	2648.92	330.83
4c	2810.60	126.03	2717.17	263.58	2760.42	203.01
4d	2786.58	41.44	2782.60	64.15	2789.72	35.96
4e	2185.35	27.87	2176.72	49.41	2183.09	24.06
4f	2446.00	99.88	2260.25	370.01	2306.14	327.96

C.5 Detailed Recall-Precision Curve Results

Tables C.10 and C.11 show the single precision values from the recall-precision curves presented in Figures 3.7–3.9. Every row shows the macro-averaged precision (mean) over all queries for a particular variant of LOG4SWS.KOM (Table C.10) or COV4SWS.KOM (Table C.11). The corresponding recall levels are depicted in the respective header row.

Table C.10: Detailed Recall-Precision Curve Results for LOG4SWS.KOM

#	0.0	0.053	0.105	0.158	0.211	0.263	0.316	0.368	0.421	0.474	0.526	0.579	0.632	0.684	0.737	0.789	0.842	0.895	0.947	1.0
1a	0.931	0.925	0.897	0.880	0.880	0.863	0.830	0.801	0.791	0.761	0.726	0.696	0.652	0.591	0.543	0.508	0.467	0.441	0.384	0.346
1b	0.931	0.925	0.897	0.880	0.880	0.867	0.818	0.800	0.787	0.760	0.724	0.696	0.661	0.619	0.587	0.548	0.484	0.457	0.388	0.350
1c	0.931	0.925	0.897	0.880	0.880	0.862	0.831	0.810	0.802	0.769	0.736	0.708	0.664	0.619	0.599	0.556	0.510	0.472	0.419	0.369
1d	0.936	0.930	0.887	0.884	0.876	0.848	0.833	0.806	0.791	0.767	0.746	0.726	0.715	0.686	0.670	0.655	0.627	0.539	0.462	0.431
2a	0.995	0.993	0.988	0.960	0.955	0.912	0.880	0.827	0.812	0.767	0.728	0.699	0.661	0.609	0.559	0.515	0.498	0.472	0.414	0.376
2b	0.995	0.993	0.988	0.961	0.955	0.914	0.880	0.826	0.811	0.778	0.730	0.709	0.665	0.614	0.562	0.517	0.483	0.460	0.406	0.360
2c	0.995	0.992	0.988	0.961	0.957	0.910	0.882	0.837	0.825	0.784	0.742	0.711	0.676	0.631	0.581	0.537	0.514	0.487	0.421	0.389
2d	0.990	0.990	0.988	0.975	0.940	0.935	0.864	0.847	0.823	0.763	0.720	0.685	0.668	0.656	0.616	0.582	0.561	0.518	0.453	0.430
3a	0.990	0.990	0.978	0.948	0.941	0.914	0.879	0.814	0.786	0.731	0.710	0.660	0.645	0.612	0.585	0.512	0.495	0.457	0.404	0.360
3b	0.987	0.985	0.978	0.952	0.942	0.915	0.876	0.836	0.797	0.748	0.720	0.675	0.651	0.614	0.599	0.519	0.497	0.460	0.404	0.355
3c	0.987	0.985	0.983	0.952	0.940	0.918	0.882	0.856	0.808	0.767	0.731	0.683	0.663	0.620	0.604	0.527	0.507	0.470	0.420	0.384
3d	0.975	0.975	0.972	0.948	0.917	0.900	0.847	0.809	0.784	0.741	0.733	0.679	0.657	0.639	0.618	0.551	0.529	0.486	0.452	0.414

Table C.11: Detailed Recall-Precision Curve Results for COV4SWS.KOM

#	0.0	0.053	0.105	0.158	0.211	0.263	0.316	0.368	0.421	0.474	0.526	0.579	0.632	0.684	0.737	0.789	0.842	0.895	0.947	1.0
1a	0.862	0.806	0.787	0.768	0.763	0.757	0.742	0.724	0.701	0.688	0.677	0.654	0.628	0.607	0.570	0.519	0.430	0.388	0.348	0.316
1b	0.855	0.819	0.797	0.772	0.751	0.734	0.729	0.714	0.694	0.684	0.636	0.611	0.579	0.577	0.533	0.476	0.388	0.310	0.260	0.249
1c	0.822	0.789	0.778	0.778	0.771	0.771	0.766	0.746	0.742	0.713	0.678	0.635	0.608	0.596	0.561	0.480	0.403	0.336	0.284	0.257
1d	0.852	0.799	0.794	0.769	0.769	0.760	0.747	0.735	0.719	0.698	0.658	0.647	0.591	0.588	0.560	0.470	0.369	0.294	0.254	0.242
1e	0.931	0.925	0.898	0.883	0.881	0.842	0.833	0.802	0.782	0.760	0.713	0.657	0.630	0.588	0.570	0.534	0.507	0.458	0.392	0.355
1f	0.931	0.922	0.890	0.871	0.871	0.857	0.831	0.820	0.809	0.789	0.752	0.699	0.663	0.601	0.579	0.536	0.479	0.438	0.413	0.398
2a	0.896	0.865	0.856	0.831	0.829	0.821	0.800	0.794	0.771	0.758	0.721	0.705	0.678	0.658	0.606	0.533	0.464	0.423	0.383	0.341
2b	0.871	0.861	0.844	0.811	0.807	0.794	0.783	0.783	0.779	0.779	0.692	0.665	0.616	0.610	0.567	0.498	0.423	0.344	0.292	0.265
2c	0.962	0.962	0.956	0.925	0.889	0.868	0.848	0.803	0.743	0.708	0.687	0.662	0.641	0.625	0.594	0.520	0.462	0.419	0.351	0.311
2d	0.942	0.942	0.942	0.912	0.905	0.890	0.853	0.822	0.786	0.750	0.704	0.675	0.628	0.608	0.564	0.510	0.444	0.367	0.329	0.286
2e	0.995	0.993	0.988	0.971	0.956	0.897	0.868	0.829	0.811	0.787	0.743	0.683	0.661	0.628	0.588	0.550	0.508	0.470	0.417	0.379
2f	0.990	0.989	0.984	0.956	0.951	0.908	0.875	0.845	0.836	0.796	0.761	0.700	0.654	0.613	0.586	0.540	0.526	0.478	0.415	0.386
3a	0.899	0.871	0.860	0.836	0.831	0.826	0.812	0.808	0.780	0.768	0.727	0.701	0.676	0.659	0.614	0.544	0.469	0.434	0.390	0.356
3b	0.874	0.864	0.846	0.817	0.806	0.800	0.792	0.785	0.784	0.784	0.703	0.682	0.619	0.608	0.569	0.505	0.430	0.350	0.300	0.271
3c	0.976	0.973	0.949	0.925	0.892	0.875	0.848	0.811	0.784	0.746	0.735	0.705	0.688	0.664	0.622	0.552	0.488	0.455	0.397	0.355
3d	0.969	0.969	0.945	0.925	0.878	0.858	0.842	0.811	0.783	0.748	0.736	0.711	0.699	0.674	0.622	0.554	0.507	0.472	0.416	0.364
3e	0.987	0.985	0.983	0.955	0.941	0.904	0.856	0.834	0.792	0.761	0.736	0.698	0.675	0.637	0.618	0.557	0.515	0.466	0.412	0.370
3f	0.985	0.985	0.978	0.944	0.938	0.905	0.864	0.830	0.801	0.751	0.732	0.670	0.642	0.608	0.600	0.516	0.509	0.463	0.398	0.344
4a	0.972	0.972	0.972	0.949	0.912	0.912	0.875	0.841	0.820	0.807	0.771	0.721	0.703	0.679	0.654	0.606	0.549	0.505	0.440	0.396
4b	0.972	0.972	0.957	0.915	0.879	0.870	0.865	0.836	0.815	0.775	0.752	0.730	0.715	0.684	0.659	0.591	0.539	0.506	0.418	0.359
4c	0.972	0.972	0.963	0.931	0.899	0.877	0.863	0.837	0.803	0.769	0.743	0.699	0.662	0.627	0.594	0.528	0.467	0.426	0.325	0.278
4d	0.976	0.976	0.971	0.940	0.893	0.882	0.868	0.845	0.793	0.764	0.733	0.704	0.675	0.634	0.598	0.543	0.486	0.438	0.334	0.279
4e	0.990	0.990	0.984	0.956	0.946	0.910	0.862	0.848	0.814	0.773	0.735	0.679	0.642	0.610	0.569	0.483	0.457	0.433	0.339	0.299
4f	0.990	0.990	0.984	0.958	0.952	0.945	0.896	0.885	0.832	0.802	0.723	0.667	0.607	0.566	0.527	0.450	0.440	0.417	0.362	0.300

D Further Details

In the following, further details, which have been omitted in Chapter 4 in order to keep the respective descriptions clearly arranged, will be presented. Section D.1 addresses the integration of SPARQL in UDDI (Section 4.3), Section D.2 addresses the integration of SWS2QL in ebXML Registry (Section D.1).

D.1 Integrating SPARQL in UDDI

Further details regarding the work presented in Section 4.3 concern the enhanced respectively newly created tModels (Section D.1.1), a SPARQL query needed at publication time (Section D.1.2), example SPARQL queries for service discovery (Section D.1.3), and regular expressions needed for query processing (Section D.1.4).

D.1.1 tModels

Listing D.1: RDF Reference tModel

```
<tModel xmlns=''urn:uddi-org:api_v2'' tModelKey=''uuid:30D91130-4EDB-11DE-9130-F42F2CB3EADE''>
   <name>RDF Reference</name>
   <description xml:lang=''en''>
      This tModel is used to let another tModel reference an RDF file.
   </description>
   <overviewDoc>
      <overviewURL>http://www.example.com/overview</overviewURL>
   </overviewDoc>
</tModel>
```

Listing D.2: RDF Categorization tModel

```
<tModel xmlns=''urn:uddi-org:api_v2'' tModelKey=''uuid:EB342380-4D87-11DE-A380-95EBB35FBB35''>
   <name>RDF Categorization</name>
   <description xml:lang=''en''>
      Specifies a tModel as pointing to an RDF description.
   </description>
   <overviewDoc>
      <description xml:lang=''en''>
         Specifies a tModel as pointing to an RDF description.
      </description>
      <overviewURL>http://www.example.com/overview</overviewURL>
   </overviewDoc>
   <categoryBag>
      <keyedReference
         tModelKey=''uuid:C1ACF26D-9672-4404-9D70-39B756E62AB4''
         keyName=''types''
         keyValue=''categorization''/>
   </categoryBag>
</tModel>
```

As it has been described in Section 4.3.3, two additional tModels have been added to the UDDI standard in order to reference RDF files from WSDL-based service descriptions. These tModels are depicted in Listings D.1 and D.2. The first tModel, *RDF Reference tModel*, is used to let another tModel reference an RDF file. The latter, *RDF Categorization tModel*, is used to categorize RDF tModels.

Listing D.4 shows the example RDF tModel for the WSDL tModel depicted in Listing D.3. An RDF tModel is generated when publishing a WSDL in UDDI using a template. The `overviewURL` (Lines 10–12) contains the path to the corresponding RDF file describing a WSDL. The filename is consistent with the tModel key of the associated WSDL tModel. Apart from this and the name (Line 2) which is derived

Listing D.3: WSDL tModel (Example)

```
1  <tModel xmlns="urn:uddi-org:api_v2" tModelKey="uuid:DDDDDDDD-DDDD-DDDD-DDDD-DDDDDDDDDDDD">
2     <name>MyTModel</name>
3     <description xml:lang=''en''>
4        The overviewURL of this tModel points to a WSDL file.
5     </description>
6     <overviewDoc>
7        <description xml:lang=''en''>WSDL file of a Web service.</description>
8        <overviewURL>http://www.example.com/services/service.wsdl</overviewURL>
9     </overviewDoc>
10    <categoryBag>
11       <keyedReference
12          tModelKey=''uuid:C1ACF26D-9672-4404-9D70-39B756E62AB4''
13          keyName=''Specification for a Web service described in WSDL''
14          keyValue=''wsdlSpec''/>
15       <keyedReference
16          tModelKey=''uuid:30D91130-4ED-11DE-9130-F42F2CB3EADE''
17          keyName=''RDF reference''
18          keyValue=''uuid:RRRRRRRR-RRRR-RRRR-RRRR-RRRRRRRRRRRR''/>
19    </categoryBag>
20 </tModel>
```

Listing D.4: RDF tModel (Example)

```
1  <tModel xmlns=''urn:uddi-org:api_v2'' tModelKey=''uuid:RRRRRRRR-RRRR-RRRR-RRRR-RRRRRRRRRRRR''>
2     <name>RDF data for tModel named MyTModel</name>
3     <description xml:lang=''en''>
4        The overviewURL of this tModel points to an RDF file which was generated out of a WSDL file. The WSDL
            file was taken from a tModel which name is saved in the name of this tModel.
5     </description>
6     <overviewDoc>
7        <description xml:lang=''en''>
8           The RDF-file created out of a WSDL file.
9        </description>
10       <overviewURL>
11          http://localhost/rdf/DDDDDDDD-DDDD-DDDD-DDDD-DDDDDDDDDDDD.rdf
12       </overviewURL>
13    </overviewDoc>
14    <categoryBag>
15       <keyedReference
16          tModelKey=''uuid:EB342380-4D87-11DE-A380-95EBB35FBB35''
17          keyName=''Specification for a RDF file''
18          keyValue=''rdfSpec''/>
19    </categoryBag>
20 </tModel>
```

from the tModel name of the associated WSDL tModel, all RDF tModels contain the same information. As explained in Appendix A.2.1, RDF tModels are categorized by a `keyedreference` pointing to the according categorization tModel shown in Listing D.2; this can be seen in Lines 15–18 of Listing D.4. In Listing D.3, Lines 15–18 show how the RDF Reference tModel is used in order to link a WSDL tModel to an RDF tModel.

D.1.2 SPARQL Queries at Publication Time

At publication time, a SPARQL query is needed in order to find the semantic concepts for caching (cp. Section 4.3.3). An example SPARQL query, which finds all semantic concepts of inputs based on its RDF representation can be seen in Listing D.5. The actual element type is indicated by the `w3rdf:type` element in Line 7. Corresponding to the mapping from SAWSDL to RDF presented in Appendix A.1.2, the following parameter values are possible: `InputMessage`, `OutputMessage`, `Interface`, and `InterfaceOperation`. Actually, it is possible to make use of all mappings defined in the WSDL to RDF mapping by Kopecký, e.g., `InterfaceFault` [143]. However, as these service components are not applied in matchmaking and service discovery, they are not regarded here.

Listing D.5: SPARQL Query Identifying all Inputs

```
PREFIX w3rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>                    1
PREFIX w3wsdlrdf: <http://www.w3.org/ns/wsdl-rdf#>                             2
PREFIX w3sawsdl: <http://www.w3.org/ns/sawsdl#>                                3
PREFIX komrdf: <http://www.kom.tu-darmstadt.de/ns/xsd-rdf#>                    4
                                                                               5
SELECT ?uri WHERE {                                                            6
   ?a w3rdf:type w3wsdlrdf:InputMessage .                                      7
   ?a w3wsdlrdf:elementDeclaration ?b .                                        8
   ?b komrdf:describedBy ?c .                                                  9
   ?c komrdf:complexType ?d .                                                  10
   ?d komrdf:sequence ?e .                                                     11
   ?e komrdf:element ?f .                                                      12
   ?f komrdf:type ?g .                                                         13
   ?g w3sawsdl:modelReference ?uri                                             14
}                                                                              15
```

D.1.3 Example SPARQL Queries

Listing D.6: SPARQL ASK Query (Example)

```
PREFIX w3rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>                    1
PREFIX w3wsdlrdf: <http://www.w3.org/ns/wsdl-rdf#>                             2
PREFIX w3sawsdl: <http://www.w3.org/ns/sawsdl#>                                3
PREFIX komrdf: <http://www.kom.tu-darmstadt.de/ns/xsd-rdf#>                    4
PREFIX books: <http://127.0.0.1/ontology/books.owl#>                           5
ASK {                                                                          6
   {                                                                           7
   ?a w3rdf:type w3wsdlrdf:InputMessage .                                      8
   ?a w3wsdlrdf:elementDeclaration ?b .                                        9
   ?b komrdf:describedBy ?c .                                                  10
   ?c komrdf:complexType ?d .                                                  11
   ?d komrdf:sequence ?e .                                                     12
   ?e komrdf:element ?f .                                                      13
   ?f komrdf:type ?g .                                                         14
   ?g w3sawsdl:modelReference books:Title                                      15
   }                                                                           16
   {                                                                           17
   ?a w3rdf:type w3wsdlrdf:OutputMessage .                                     18
   ?a w3wsdlrdf:elementDeclaration ?b .                                        19
   ?b komrdf:describedBy ?c .                                                  20
   ?c komrdf:complexType ?d .                                                  21
   ?d komrdf:sequence ?e .                                                     22
   ?e komrdf:element ?f .                                                      23
   ?f komrdf:type ?g .                                                         24
   ?g w3sawsdl:modelReference books:Novel                                      25
   }                                                                           26
}                                                                              27
```

Listings D.6 and D.7 show example SPARQL ASK queries to be used in service discovery as presented in Section 4.3. The former query shows a standard SPARQL query without any extensions – as it can be seen, the actual query specification is done using concepts blocks. In Listing D.6, two concept blocks are defined – the first concept block (Lines 7–16) specifies that the requested service needs to have an

Listing D.7: SPARQL ASK Query Extended with Minimum DoMs (Example)

```
1  PREFIX w3rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2  PREFIX w3wsdlrdf: <http://www.w3.org/ns/wsdl-rdf#>
3  PREFIX w3sawsdl: <http://www.w3.org/ns/sawsdl#>
4  PREFIX komrdf: <http://www.kom.tu-darmstadt.de/ns/xsd-rdf#>
5  PREFIX books: <http://127.0.0.1/ontology/books.owl#>
6  ASK {
7     {
8     ?a w3rdf:type w3wsdlrdf:InputMessage .
9     ?a w3wsdlrdf:elementDeclaration ?b .
10    ?b komrdf:describedBy ?c .
11    ?c komrdf:complexType ?d .
12    ?d komrdf:sequence ?e .
13    ?e komrdf:element ?f .
14    ?f komrdf:type ?g .
15    ?g w3sawsdl:modelReference books:Title.SIMILARITY.EXACT
16    }
17    {
18    ?a w3rdf:type w3wsdlrdf:OutputMessage .
19    ?a w3wsdlrdf:elementDeclaration ?b .
20    ?b komrdf:describedBy ?c .
21    ?c komrdf:complexType ?d .
22    ?d komrdf:sequence ?e .
23    ?e komrdf:element ?f .
24    ?f komrdf:type ?g .
25    ?g w3sawsdl:modelReference books:Novel.SIMILARITY.SUPER
26    }
27 }
```

input referencing the semantic concept `Title` from the books ontology (Line 5), the second concept block (Lines 17–26) specifies that the service needs to have an output referencing the semantic concept `Novel` from the same ontology.

In Listing D.7, the extension by similarity ranges is depicted. As it can be seen in Lines 15 and 25, this is done by adding the minimum DoM (i.e., threshold) to a semantic concept's URI. In the example from Listing D.7 only those services which accept a data type referencing the semantic concept `books.owl#Title` and offering the data type referencing either `books.owl#Novel` or one of its superclasses, are added to the result set.

D.1.4 Regular Expressions Needed for Query Processing

Listing D.8: Regular Expression to Identify Concept Blocks

```
\(bgp\s+?(\(triple[^\(\)]*\)\s+?)+?\s+?\)
```

Listing D.9: Regular Expression to Identify Concepts in Concept Blocks

```
1  (\(bgp.+?\Qrdf-syntax-ns#type>\E\s\<([^\(\)\<\>]+)\>.+?\Qsawsdl#modelReference>
2  \E\s\<([^\(\)\<\>]+)\>.+?\))
3  |
4  (\(bgp.+?\Qsawsdl#modelReference>\E\s\<([^\(\)\<\>]+)\>.+?\Qrdf-syntax-ns#type>
5  \E\s\<([^\(\)\<\>]+)\>.+?\))
```

In the publication process for RDF-based service descriptions in UDDI (cp. Section 4.3), regular expressions are needed in order to extract concept blocks from SPARQL queries (Listing D.8) as well as to identify the semantic concept in a concept block (Listing D.9).

D.2 Integrating SWS2QL in ebXML Registry

Further details regarding the work presented in Section 4.4 concern the XML schema of the abstract query model (Section D.2.1), the structure of SQL enhancements and statements (Section D.2.2), and an overview of the classes of the matchmaker interface implemented for freebXML (Section D.2.3).

D.2.1 Query Model XML Schema

The corresponding XML schema of the query model applied in Section 4.4 is shown in Figure D.1. Except for the `name` attribute of the `queryType` and the `id` and `objectRef` attributes of the `resultType`, all other attributes of the five depicted components are optional. In addition, no combination rules are explicitly stated for the logical interconnection of query container and query section elements. Instead, the specification of optional attributes referring to solution modifiers and thresholds, as well as the definition of combination rules is left to the designer of a specific query language, since the query model is intended to serve as a foundation for the specification or assessment of service queries, i.e., it can be adapted to the requirements for designing an individual query language or referred to the particularities of an existing query language. Concerning the cardinalities of the elements, the XML schema of the query model has one particularity, which needs to be mentioned. The global `queryContainer` may either contain one or more `querySection` elements or two or more `queryContainer` elements. The cardinality of the nested `queryContainer` elements has been set to two in order to avoid the nesting of single `queryContainer` elements that can easily be replaced by a single `querySection` within the global `queryContainer`.

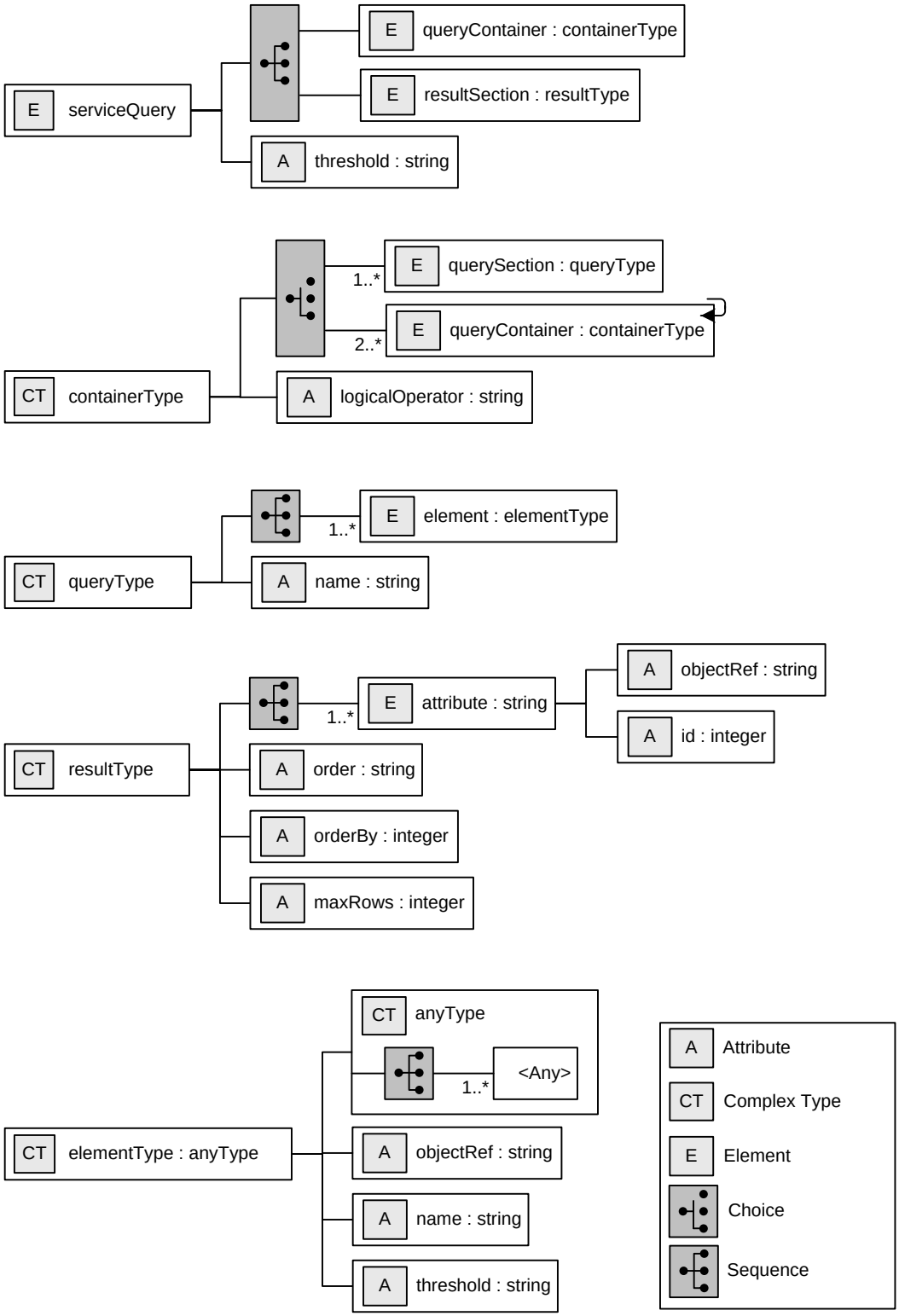


Figure D.1: Elements of a Service Query

Listing D.10: Structure of the SQL Enhancement for SWS2QL

```
AND SEMANTICS(<QueryType>(<Parameter 1>,...,<Parameter n>),<Matchmaker>)
```

Listing D.11: Structure of "Lightservice" SQL Query Statements

```
<QueryType>((<MatchingLevel>,<SemanticReference>,<DoM>),<Matchmaker>)
LIGHT((IFACE,http://www.example.com/myOnt.owl#IfaceConcept,EXACT),DEFAULT)
```

Listing D.12: Structure of "Fullservice" SQL Query Statements

```
<QueryType>((<accessURI>,<DoM>),<Matchmaker>)
FULL((http://www.example.com/services/MyWebService,EXACT),LOG4SWS.KOM)
```

Listing D.10 shows the generic structure of the SQL enhancement needed to integrate service description-based query parameters from SWS2QL in freebXML-SQL.

The native part of an SQL query is augmented by the enhanced part, which is indicated by the keywords `AND SEMANTICS` followed by the actual query statement. The structure of a query statement comprises three parts: a literal constant `LIGHT` or `FULL` denoting the respective query type, a list of the corresponding query parameters, and a further literal constant identifying the matchmaker, which should be used for the processing of the SQL enhancement. In doing so, the lightservice and fullservice query types are then established as depicted in the Listings D.11 and D.12.

Figure D.2: Registry Enhancements for the Integration of Matchmakers

For the integration of different matchmakers, the query manager implementation of freebXML has been modified and additional classes have been created. An overview of the classes is depicted in Figure D.2. Possible matchmakers can be registered within the `IMatchmakerConfig` class in the form of a string-based constant that is associated with the fully qualified name of the main class of a matchmaker. Using the Java Reflection API[1], the query manager implementation is able to redirect a query to the desired matchmaker. For the integration of a new matchmaker, the `IMatchmaker` interface can be implemented by the respective class. In doing so, two methods have to be realized by the new matchmaker; one for processing the SQL query extensions and a second method to return the optional response slot list, which can be used to send back additional information of a matching service to the client. For the proof of concept implementation presented in Section 4.4, the `LOG4SWSMatchmaker` class has been created, which establishes the connection to the matching facilities provided by LOG4SWS.KOM and defines additional methods for the processing of the fullservice and the lightservice queries.

[1] `http://java.sun.com/docs/books/tutorial/reflect/index.html`

E Author's Publications

E.1 Main Publications

[1] Stefan Schulte, Ulrich Lampe, Julian Eckert, and Ralf Steinmetz. LOG4SWS.KOM: Self-Adapting Semantic Web Service Discovery for SAWSDL. In *IEEE 2010 Fourth International Workshop of Software Engineering for Adaptive Service-Oriented Systems (SEASS '10) at 2010 IEEE 6th World Congress on Services (SERVICES 2010)*, pages 511–518. IEEE Computer Society, Washington, DC, USA, 2010.

[2] Stefan Schulte, Melanie Siebenhaar, Julian Eckert, and Ralf Steinmetz. Query Languages for Semantic Web Services. In *Forthcoming in: Informatik 2010*. Gesellschaft für Informatik, 2010.

[3] Stefan Schulte, Kay Kadner, Nicolas Repp, and Ralf Steinmetz. Applied Service Engineering for Single Services and Corresponding Service Landscapes. In *15th Americas Conference on Information Systems (AMCIS 2009)*. Association for Information Systems, Atlanta, GA, USA, 2009.

[4] Stefan Schulte, Nicolas Repp, Dieter Schuller, and Ralf Steinmetz. From Web Service Policies to Automatic Deviation Handling: Supporting Semantic Description of Reactions to Policy Violations. In *Third IEEE International Conference on Semantic Computing (ICSC 2009)*, pages 312–317. IEEE Computer Society, Washington, DC, USA, 2009.

[5] Stefan Schulte. An Approach to Evaluate and Enhance the Retrieval of Web Services Based on Semantic Information. In *Ph.D. Symposium at the 5th European Semantic Web Conference (ESWC 2008)*, volume 358 of *CEUR Workshop Proceedings*, pages 61–65. CEUR-WS.org, 2008.

[6] Stefan Schulte, Julian Eckert, Nicolas Repp, and Ralf Steinmetz. An Approach to Evaluate and Enhance the Retrieval of Semantic Web Services. In *5th International Conference on Service Systems and Service Management (ICSSSM'08)*, pages 237–243. IEEE Computer Society, Washington, DC, USA, 2008.

[7] Stefan Schulte, Nicolas Repp, Julian Eckert, Rainer Berbner, Korbinian von Blanckenburg, Ralf Schaarschmidt, and Ralf Steinmetz. General Requirements of Banks on IT Architectures and the Service-Oriented Architecture Paradigm. In *Enterprise Applications and Services in the Finance Industry*, volume 4 of *Lecture Notes in Business Information Processing*, pages 66–80. Springer, Heidelberg Berlin, 2008.

[8] Stefan Schulte, Nicolas Repp, Julian Eckert, Rainer Berbner, Korbinian von Blanckenburg, Ralf Schaarschmidt, and Ralf Steinmetz. Potential Risks and Benefits of Service-Oriented Collaboration – Basic Considerations and Results from an Empirical Study. In *Second IEEE International Conference on Digital Ecosystems and Technologies (DEST 2008)*, pages 155–160. IEEE Computer Society, Washington, DC, USA, 2008.

[9] Stefan Schulte, Nicolas Repp, Julian Eckert, Ralf Steinmetz, Korbinian von Blanckenburg, and Ralf Schaarschmidt. Service-oriented Architectures – Status Quo and Prospects for the German Banking Industry. *Interoperability in Business Information Systems (IBIS)*, 3(2):9–31, 2008.

[10] Stefan Schulte, Rainer Berbner, Ralf Steinmetz, and Mathias Uslar. Implementing and Evaluating the Common Information Model in a Relational and RDF-based Database. In *Third International ICSC Symposium on Information Technologies in Environmental Engineering (ITEE 2007)*, Environmental Science and Engineering, pages 109–118. Springer, Berlin Heidelberg, 2007.

[11] Stefan Schulte, Nicolas Repp, Rainer Berbner, Ralf Steinmetz, and Ralf Schaarschmidt. Service-Oriented Architecture Paradigm: Major Trend or Hype for the German Banking Industry? In *13th Americas Conference on Information Systems (AMCIS 2007)*. Association for Information Systems, Atlanta, GA, USA, 2007.

[12] Stefan Schulte, Nicolas Repp, Julian Eckert, Rainer Berbner, Ralf Steinmetz, and Ralf Schaarschmidt. Familiarity with and Usage of Service-oriented Architectures in German Banks. *EFL Quarterly*, (2):4–5, 2007.

[13] Stefan Schulte, Nicolas Repp, Ralf Schaarschmidt, Julian Eckert, Rainer Berbner, and Ralf Steinmetz. Potentielle Auswirkungen des Paradigmas der Service-orientierten Architekturen auf die Softwarebranche – Ergebnisse einer Studie aus der deutschen Bankenindustrie. In *Science Meets Business (Tagungsband des Stuttgarter Softwaretechnik Forum 2007)*, pages 21–30. Fraunhofer IRB Verlag, Stuttgart, 2007.

[14] Stefan Schulte, Nicolas Repp, Ralf Schaarschmidt, Julian Eckert, Rainer Berbner, Ralf Steinmetz, and Korbinian von Blanckenburg. *Service-orientierte Architekturen – Status quo und Perspektive für die deutsche Bankenbranche*. Books on Demand GmbH, Norderstedt, 2007.

E.2 Other Publications

[1] Apostolos Papageorgiou, Tronje Krop, Sebastian Ahlfeld, Stefan Schulte, Julian Eckert, and Ralf Steinmetz. Enhancing Availability with Self-Organization Extensions in a SOA Platform. In *Fifth International Conference on Internet and Web Applications and Services (ICIW 2010)*, pages 161–166. 2010.

[2] Dieter Schuller, Julian Eckert, André Miede, Stefan Schulte, and Ralf Steinmetz. QoS-Aware Service Composition for Complex Workflows. In *Fifth International Conference on Internet and Web Applications and Services (ICIW 2010)*, pages 333–338. 2010.

[3] Apostolos Papageorgiou, Stefan Schulte, Dieter Schuller, Michael Niemann, Nicolas Repp, and Ralf Steinmetz. Governance of a Service-Oriented Architecture for Environmental and Public Security. In *Fourth International ICSC Symposium on Information Technologies in Environmental Engineering (ITEE 2009)*, Environmental Science and Engineering, pages 109–118. Springer, Heidelberg Berlin, 2009.

[4] Nicolas Repp, Stefan Schulte, and Ulrike Steffens. IT-Governance in verteilten Systemen – Vorwort der Workshop-Leitung. In *Informatik 2009: Im Focus das Leben, Beiträge der 39. Jahrestagung der Gesellschaft füur Informatik e.V. (GI), 28.9.-2.10.2009, Lübeck, Proceedings*, volume 154 of *Lecture Notes in Informatics*, pages 474–476. Gesellschaft für Informatik e. V., Bonn, 2009.

[5] Dieter Schuller, Apostolos Papageorgiou, Stefan Schulte, Julian Eckert, Nicolas Repp, and Ralf Steinmetz. Process Reliability in Service-Oriented Architectures. In *Third IEEE International Conference on Digital Ecosystems and Technologies (DEST 2009)*, pages 640–645. IEEE Computer Society, Washington, DC, USA, 2009.

[6] Rainer Berbner, Karsten Meister, Tanja Schmedes, and Stefan Schulte. Serviceorientierte Architekturen. In *Handbuch der Software-Architektur, 2. Auflage*, chapter 23, pages 469–488. dpunkt.verlag, Heidelberg, 2008.

[7] Julian Eckert, Stefan Schulte, Michael Niemann, Nicolas Repp, and Ralf Steinmetz. Worst-Case Workflow Performance Optimization. In *Third International Conference on Internet and Web Applications and Services (ICIW 2008)*, pages 632–637. IEEE Computer Society, Washington, DC, USA, 2008.

[8] Julian Eckert, Stefan Schulte, Nicolas Repp, Rainer Berbner, and Ralf Steinmetz. Queuing-based Capacity Planning Approach for Web Service Workflows Using Optimization Algorithms. In *Second IEEE International Conference on Digital Ecosystems and Technologies (DEST 2008)*, pages 313–318. IEEE Computer Society, Washington, DC, USA, 2008.

[9] André Miede, Michael Niemann, Stefan Schulte, Julian Eckert, Aneta Kabzeva, Nicolas Repp, and Ralf Steinmetz. Selected Topics in Service Engineering and Management for Enterprise Systems. In *Inaugural Workshop on Enterprise Systems Research in MIS (Pre-ICIS 2008)*. 2008.

[10] Michael Niemann, Melanie Siebenhaar, Julian Eckert, Stefan Schulte, Nicolas Repp, and Ralf Steinmetz. SOA in Practice. Technical Report KOM-TR-2008-04, Multimedia Communications Lab, Technische Universität Darmstadt, 2008.

[11] Nicolas Repp, Julian Eckert, Stefan Schulte, Michael Niemann, Rainer Berbner, and Ralf Steinmetz. Towards Automated Monitoring and Alignment of Service-based Workflows. In *Second IEEE International Conference on Digital Ecosystems and Technologies (DEST 2008)*, pages 235–240. IEEE Computer Society, Washington, DC, USA, 2008.

[12] Mathias Uslar, Sebastian Rohjans, Stefan Schulte, and Ralf Steinmetz. Building the Semantic Utility with Standards and Semantic Web Services. In *4th International Workshop On Semantic Web & Web Semantics (SWWS'08) at On the Move to Meaningful Internet Systems (OTM 2008)*, volume 5333 of *Lecture Notes on Computer Science*, pages 1026–1035. Springer, Berlin Heidelberg, 2008.

[13] Julian Eckert, Nicolas Repp, Stefan Schulte, Rainer Berbner, and Ralf Steinmetz. An Approach for Capacity Planning for Web Service Workflows. In *13th Americas Conference on Information Systems (AMCIS 2007)*. Association for Information Systems, Atlanta, GA, USA, 2007.

[14] Nicolas Repp, Rainer Berbner, Julian Eckert, Oliver Heckmann, Stefan Schulte, and Ralf Steinmetz. Der Einfluß von Transportschicht-Anomalien auf die Performanz von Web Services. In *15. Fachtagung Kommunikation in Verteilten Systemen (KiVS 2007)*, Informatik aktuell, pages 117–122. Springer, Berlin Heidelberg, 2007.

[15] Nicolas Repp, Stefan Schulte, Julian Eckert, Rainer Berbner, and Ralf Steinmetz. An Approach to the Analysis and Evaluation of an Enterprise Service Ecosystem. In *First International Workshop on Architectures, Concepts and Technologies for Service Oriented Computing (ACT4SOC 2007) at Second International Conference on Software and Data Technologies (ICSOFT 2007)*, pages 42–51. INSTICC Press, 2007.

[16] Nicolas Repp, Stefan Schulte, Julian Eckert, Rainer Berbner, and Ralf Steinmetz. Service-Inventur: Aufnahme und Bewertung eines Services-Bestands. In *Workshop MDD, SOA und IT-Management (MSI 2007)*, pages 13–22. GITO Verlag, Berlin, 2007.

[17] Ralf Schaarschmidt and Stefan Schulte. Studie: Banken zeigen sich offen gegenüber Service-orientierten Architekturen. *Zeitschrift für das gesamte Kreditwesen*, (3):21, 2007.

[18] Julian Eckert, Stefan Schulte, Oliver Heckmann, and Ralf Steinmetz. Decentralized Workflows. Technical Report KOM-TR-2008-04, Multimedia Communications Lab, Technische Universität Darmstadt, 2006.

F Curriculum Vitae

PERSONAL

Name	Stefan Schulte
Date of Birth	July 13th, 1980
Place of Birth	Meppen
Nationality	German

EDUCATION

Since 06/2006	Doctoral candidate at the Department of Electrical Engineering and Information Technology, Technische Universität Darmstadt
02/2005–11/2005	Studies of Information Technology at University of Newcastle, New South Wales Degree: Master of Information Technology
10/2001–08/2006	Studies of Computer Science (Major: Information Systems) at Carl von Ossietzky Universität, Oldenburg Degree: Bachelor in Computer Science
10/2000–03/2006	Studies of Economics (Major: Computer Science) at Carl von Ossietzky Universität, Oldenburg Degree: Diploma in Economics
06/1999	Abitur at Gymnasium Marianum, Meppen

WORK EXPERIENCE

Since 10/2009	Deputy head of the research group IT-Architectures at the Multimedia Communications Lab, Technische Universität Darmstadt
Since 10/2009	Deputy head of the SOA Competence Center at Hessisches Telemedia Technologie Kompetenz-Center, Darmstadt
Since 07/2009	Project manager Green Mobility at the SOA Competence Center at Hessisches Telemedia Technologie Kompetenz-Center, Darmstadt
Since 06/2006	Research assistant in the research group IT-Architectures at the Multimedia Communications Lab, Technische Universität Darmstadt
01/2006–04/2006	Student assistant in the department Business Information and Knowledge Management at OFFIS, Oldenburg
10/2004–02/2005	Internship in the department Quality Assurance (Electronics) at Volkswagen AG, Wolfsburg
07/2003–06/2004	Student assistant in the department Multimedia and Internet Information Services at OFFIS, Oldenburg

01/2001–06/2003	Student assistant in the department Business Information and Knowledge Management at OFFIS, Oldenburg

TEACHING ACTIVITIES

Since WS 07/08	Selected topics in lecture Communication Networks II at Technische Universität Darmstadt
Since WS 07/08	Organization and supervision of lecture Distributed Multimedia Systems: Ubiquitous Computing in Geschäftsprozessen (MM I) at Technische Universität Darmstadt
Since WS 2006	Tutor for various Bachelor and Master theses at Technische Universität Darmstadt
Since WS 2006	Supervision of student papers in project seminars and lab exercises at Technische Universität Darmstadt
SS 2004	Student assistant and tutor of lecture Software Engineering at Carl von Ossietzky Universität, Oldenburg

HONORS

05/2010	Best Paper Award at The Fifth International Conference on Internet and Web Applications and Services for the paper *Enhancing Availability with Self-Organization Extensions in a SOA Platform*
02/2008	Best Presentation Award at the IEEE International Conference on Digital Ecosystems and Technologies 2008 for the paper *Towards Automated Monitoring and Alignment of Service-based Workflows*
02/2008	Best Presentation Award at the IEEE International Conference on Digital Ecosystems and Technologies 2008 for the paper *Queuing-based Capacity Planning Approach for Web Service Workflows Using Optimization Algorithms*
12/2006	Master of Information Technology *with Merit* at University of Newcastle, New South Wales
04/2004	Student award in *e-Logistics Competition* of Logistikinitiative Niedersachsen for project work conducted at Carl von Ossietzky Universität, Oldenburg.

SCIENTIFIC ACTIVITIES

TPC Member	Fourth International Workshop on Service Matchmaking and Resource Retrieval in the Semantic Web (SMR2 2010), PC Member, 2010 Extended Semantic Web Conferences (ESWC), PC Member, 2010 Americas Conference on Information Systems (AMCIS), Minitrack Chair “IT-Governance, Alignment, and Architectures”, 2010 Jahrestagung der Gesellschaft für Informatik, Workshop “IT-Governance in verteilten Systemen (GVS)”, Workshop Chair, 2009, 2010 International Conference on Internet and Web Applications and Services (ICIW), TPC Member, 2009, 2010 IEEE International Conference on Digital Ecosystems and Technologies (DEST), PC Member, 2009, 2010 IEEE International Conference on Digital Ecosystems and Technologies (DEST), Track “Governance in Large Heterogeneous IT Systems”,

	Track Chair, 2009 Kommunikation in verteilten Systemen (KiVS), Workshop “Service-oriented Computing”, PC Member, 2009
Reviewer	EFL Quarterly Handbuch der Software-Architektur (2nd edition) IEEE Transactions on Industrial Electronics International Journal of Interoperability in Business Information Systems IEEE International Conference on Industrial Technology (ICIT), 2008, 2009 Americas Conference on Information Systems (AMCIS), 2007, 2009

MEMBERSHIPS

AIS	Association for Information Systems
GI	Gesellschaft für Informatik e. V.
IEEE	Institute of Electrical and Electronics Engineers
KOM-Fördergesellschaft e. V.	Gesellschaft zur Förderung von Wissenschaft und Lehre in der Informations- und Kommunikationstechnologie. Chairman since 12/2007.

G Erklärung laut §9 der Promotionsordnung

Ich versichere hiermit, dass ich die vorliegende Dissertation allein und nur unter Verwendung der angegebenen Literatur verfasst habe.

Die Arbeit hat bisher noch nicht zu Prüfungszwecken gedient.

Darmstadt, 2010